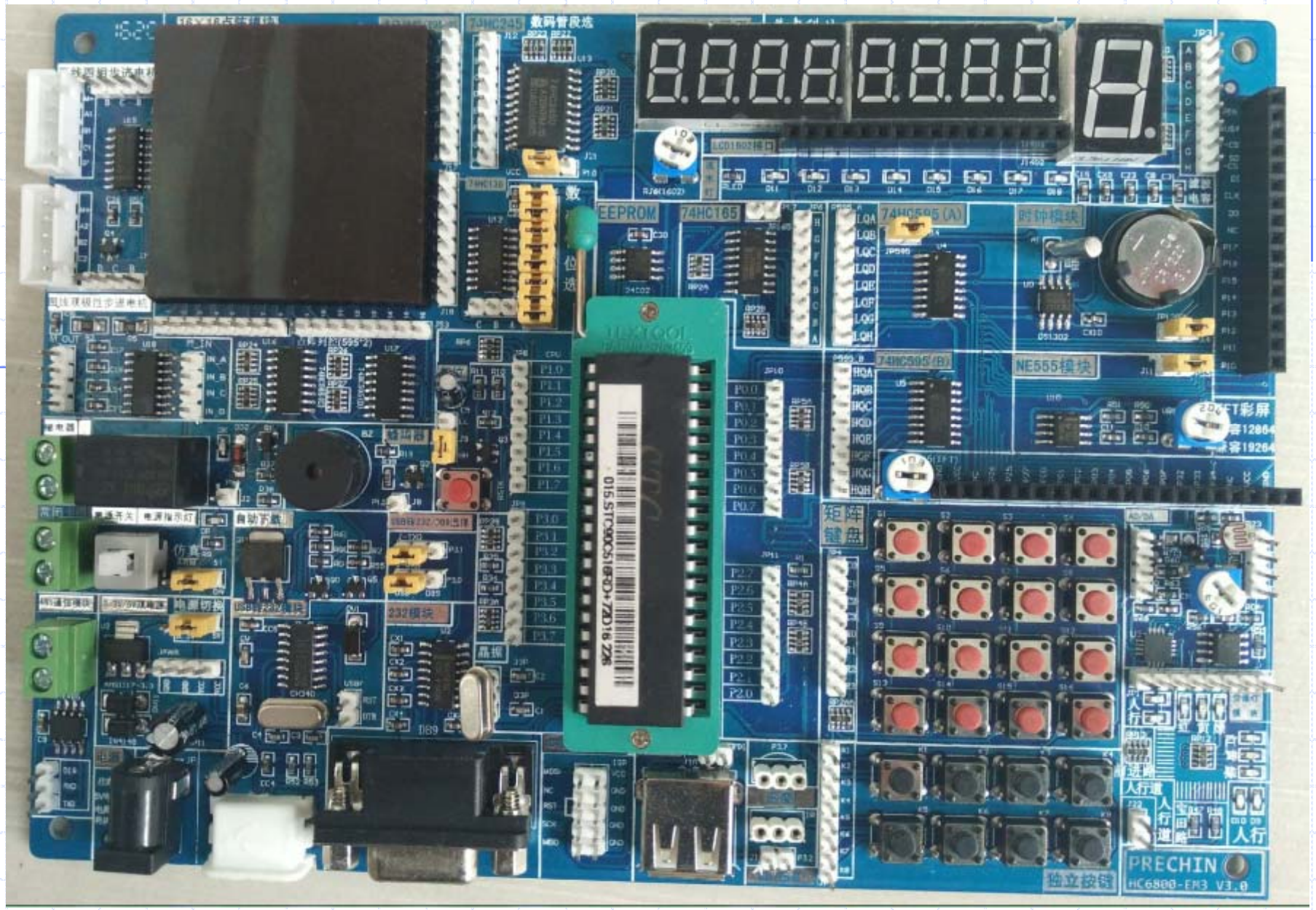
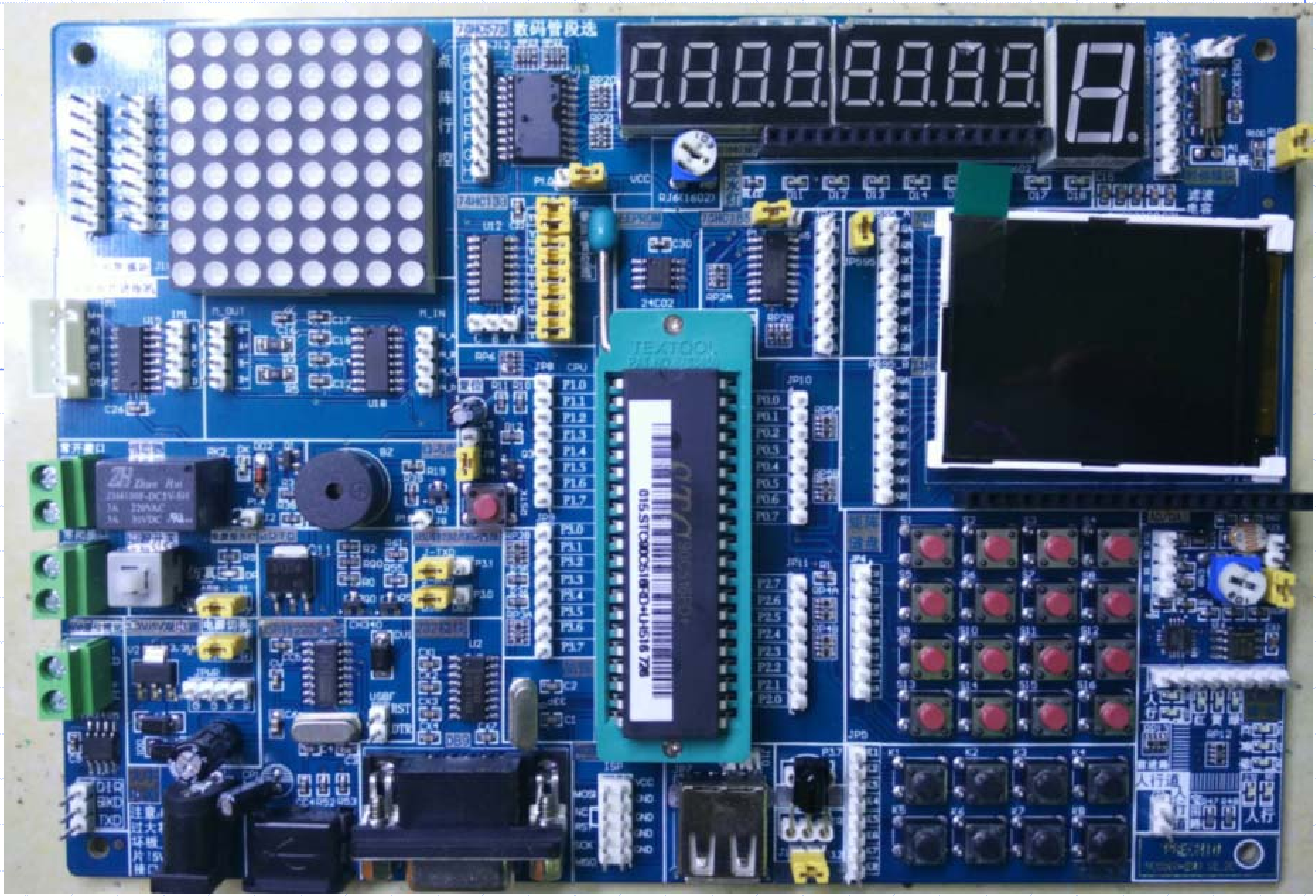
第十三讲 A/D转换和D/A转换

本讲课程：

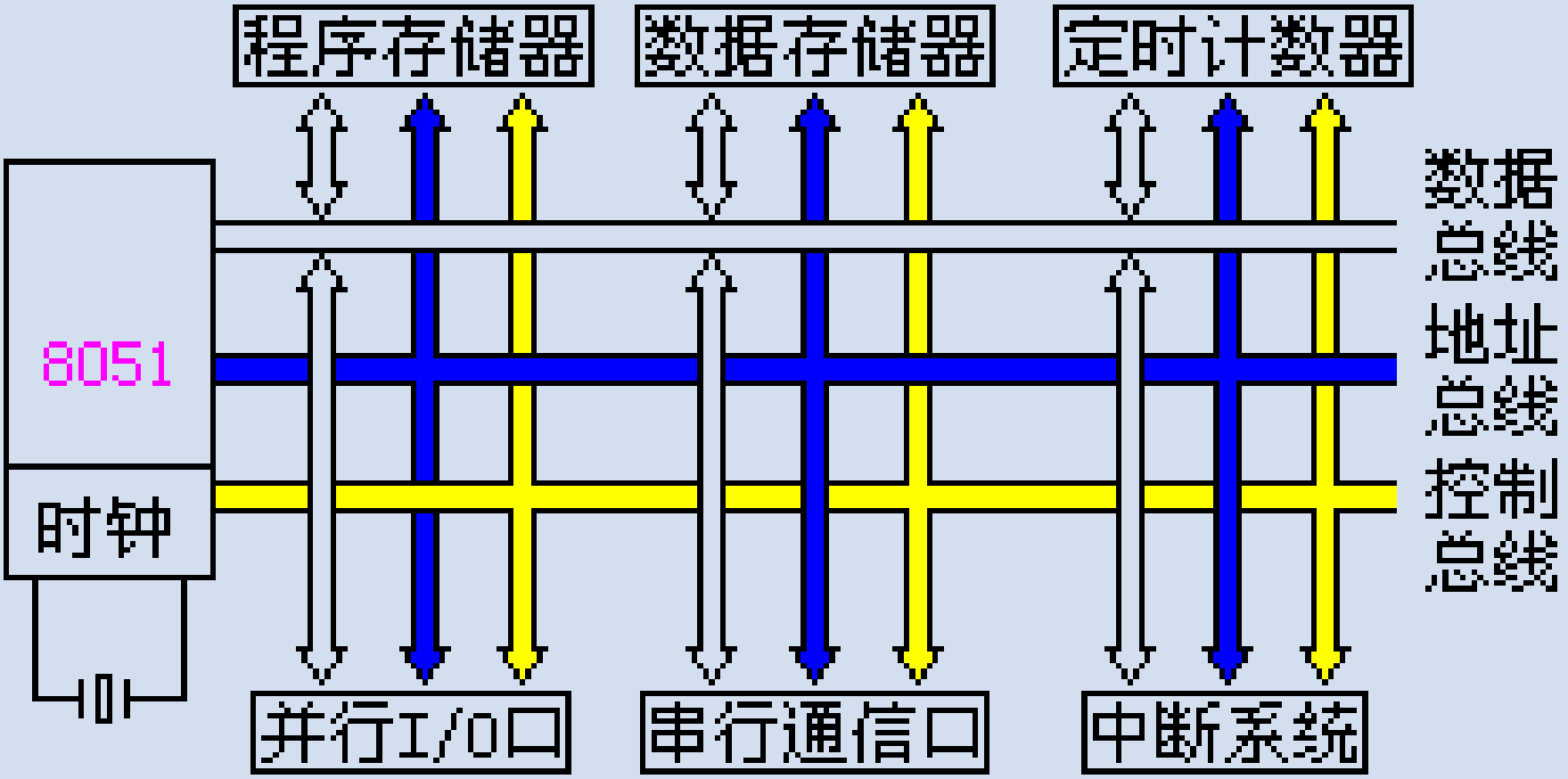
- **13.1 ARM硬件系统**
- **13.2 信号的检测**
- **13.3 A/D转换器原理**
- **13.4 MCS51的A/D转换器**
- **13.5 STM32的A/D转换器**
- **13.6 D/A转换器应用**


EEPROM
74HC165
74HC595 (A)
74HC595 (B)
时钟模块
NE555模块
矩阵键盘
独立按键
232模块
数码管段选
位选
PRECHIN
HC6800-EM3 V3.0

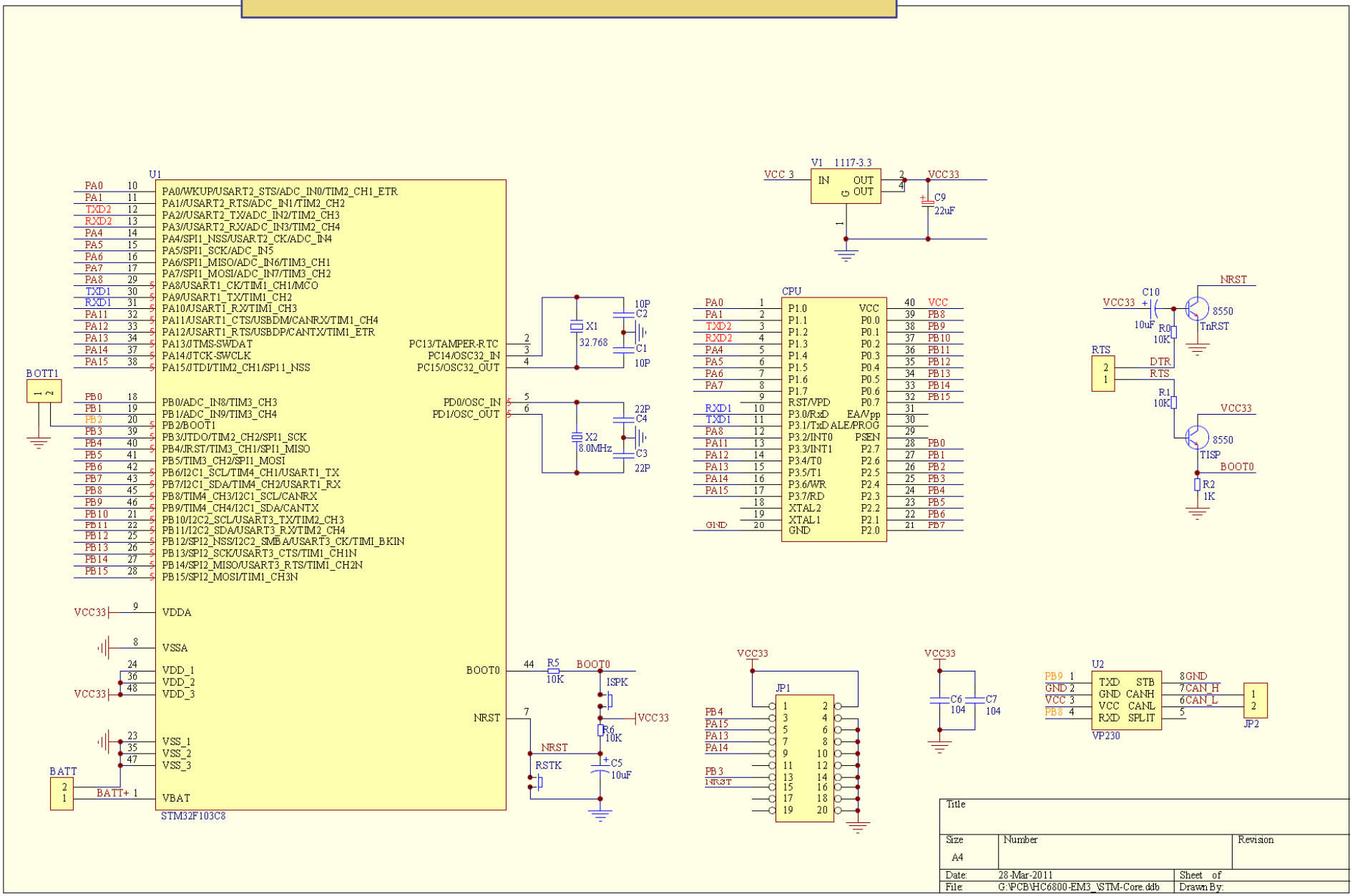


数码管段选
74HC138
24C02
TEXTOOL

8051 内部结构



◆ARM7与8051CPU

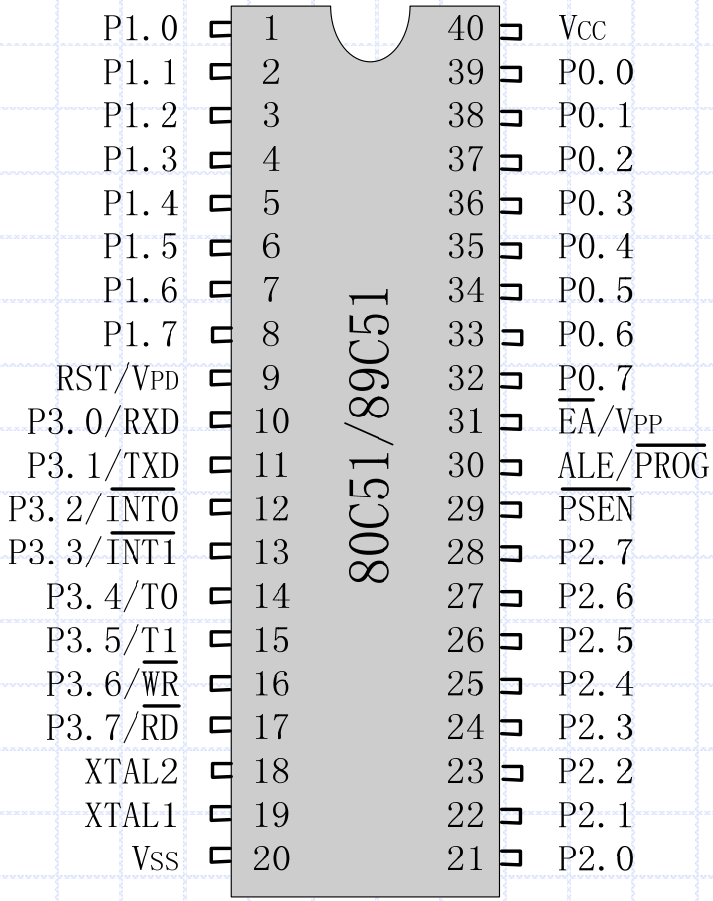


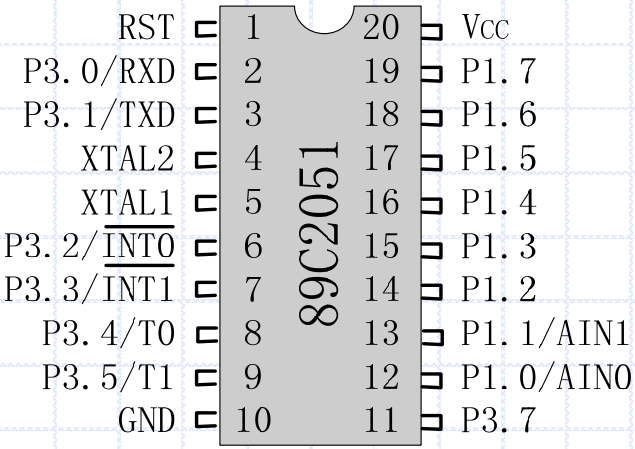

ATMEL
AT91SAM7X256
AU
0740
6R3837
A
ARM


STC
89C52RC
40C-PDIP
0826CX8101 00D
点击查看源网页


ATMEL
AT89C51
20JC
9549

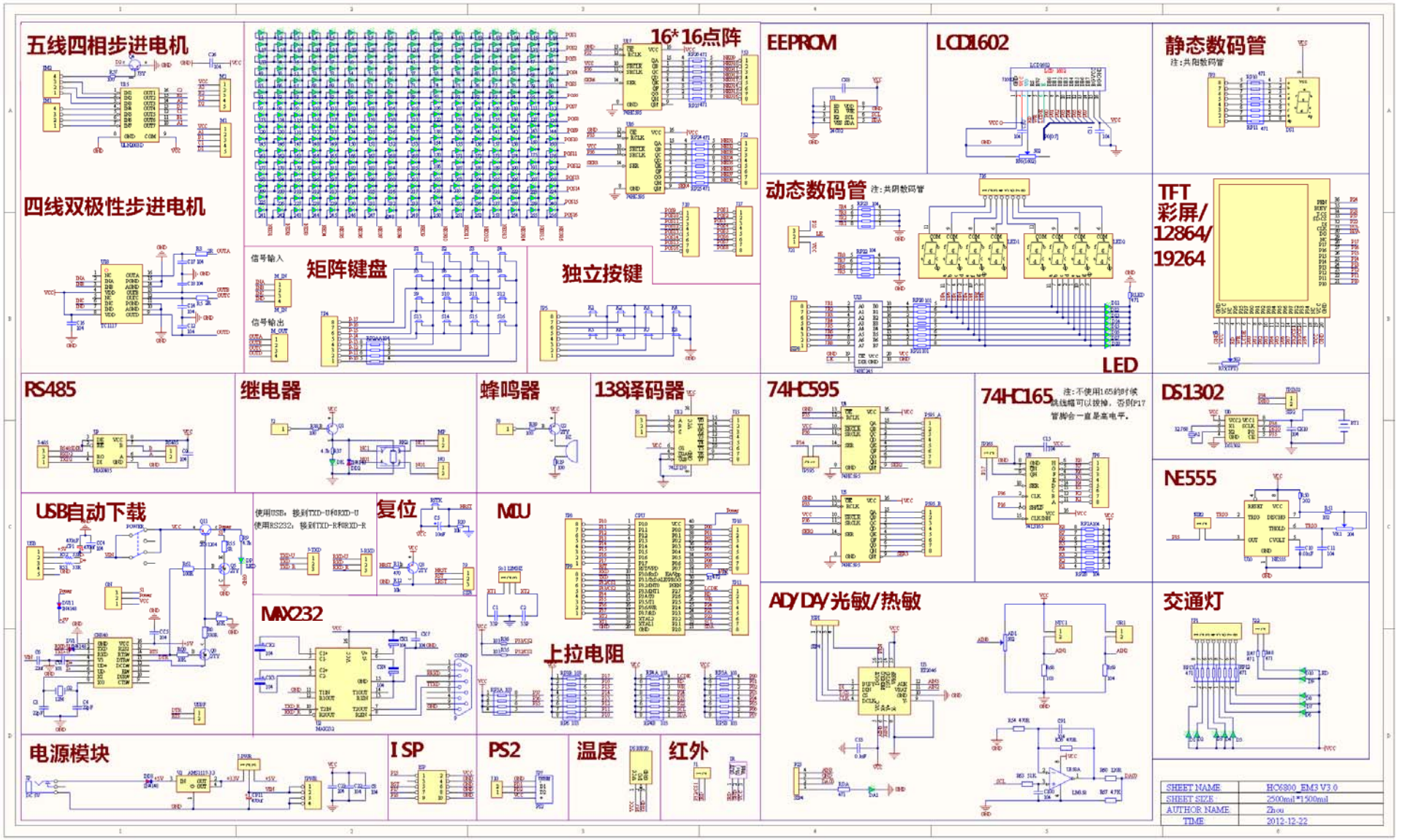
80C51的引脚封装





注：类似的还有Philips公司的
87LPC64，20引脚
8XC748/750/（751），24引脚
8X749（752），28引脚
8XC754，28引脚
等等

开放系统传感器接口


五线四相步进电机
四线双极性步进电机
16*16点阵
矩阵键盘
信号输入
信号输出
独立按键
EEPROM
LCD1602
静态数码管
注:共阳数码管
动态数码管
注:共阴数码管
LED
TFT
彩屏/
12864/
19264
RS485
继电器
蜂鸣器
138译码器
74HC595
74HC165
注:不使用165的时候
跳线帽可以拔掉，否则P17
管脚会一直是高电平。
DS1302
NE555
USB自动下载
使用USB：接到TXD-U和RXD-U
使用RS232：接到TXD-R和RXD-R
复位
MAX232
MCU
上拉电阻
AD/DA/光敏/热敏
交通灯
电源模块
ISP
PS2
温度
红外
SHEET NAME
HC6800_EM3 V3.0
SHEET SIZE
2500mil*1500mil
AUTHOR NAME
Zhou
TIME
2012-12-22

目录

本讲课程：

- 13.1 ARM硬件系统
- 13.2 信号的检测
- 13.3 A/D转换器原理
- 13.4 MCS51的A/D转换器
- 13.5 STM32的A/D转换器
- 13.6 D/A转换器应用

概述

- ARM微处理器构成一个数据采集系统或过程控制系统时，采集的外部信号与被控对象的参数通常是一些模拟量，必须把这些模拟量转换为数字量，微处理器才能接收处理；
- 微处理器的处理结果是数字量，必须将数字信号再转换为模拟信号，才能控制被控对象。
- 将模拟量转换为数字量的过程称为模数（A/D）转换，完成这一转换的器件称为模数转换器（简称ADC）；
- 将数字量转换为模拟量的过程称为数模（D/A）转换，完成这一转换的器件称为数模转换器（简称DAC）。

◆输入/输出通道概述

1、输入通道（前向通道）

被测对象与单片机联系的信号通道。包括传感器或敏感元件、通道结构、信号调节、**A/D转换**、电源的配置、抗干扰等。

2、输出通道（后向通道）

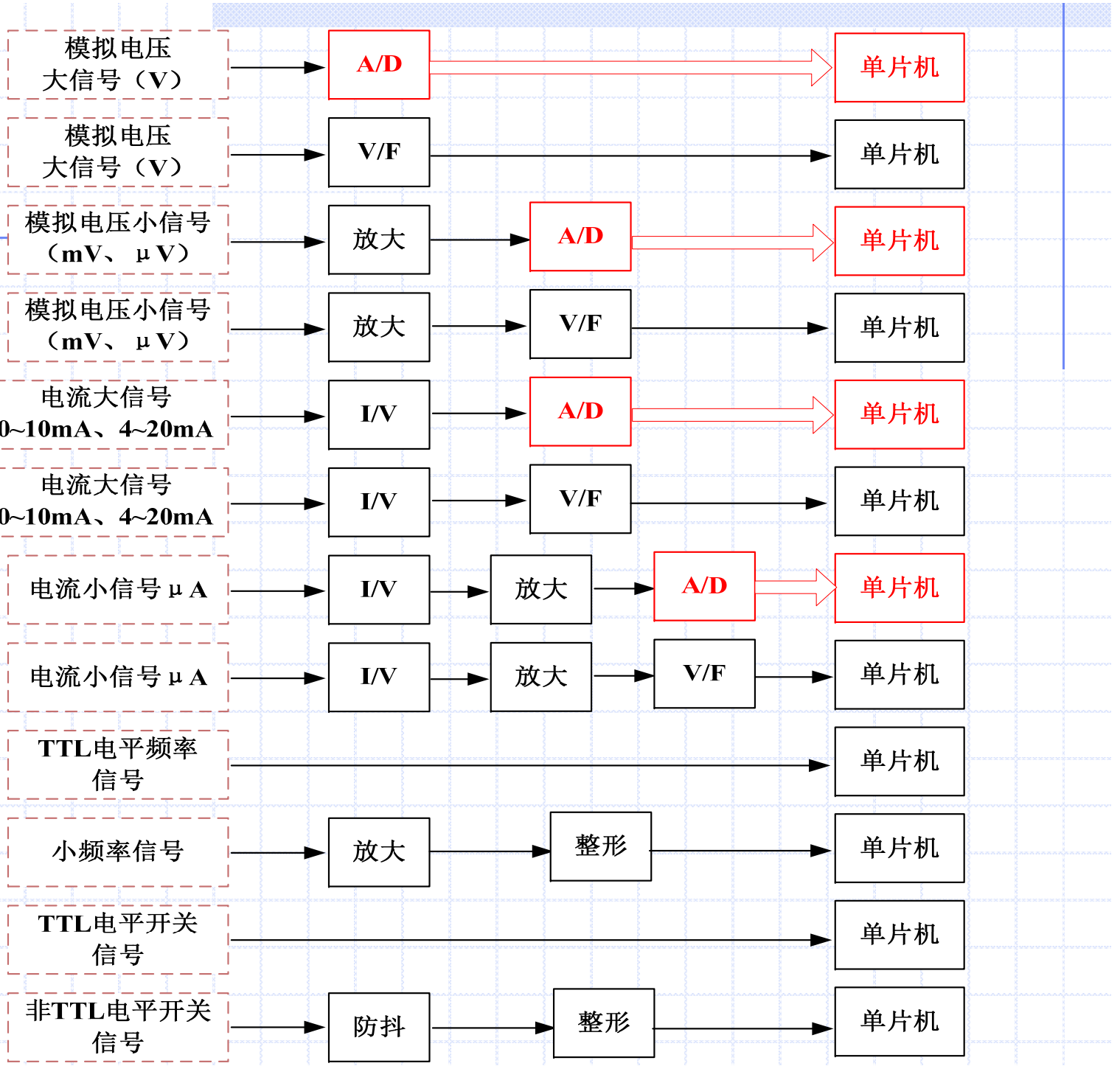
单片机与被控对象联系的信号通道。包括功率驱动、干扰的抑制、**D/A转换**等。

3、传感器输出信号形式

模拟量：模拟电压大信号（**V**）、模拟电压小信号（**mV**、**μV**）、电流大信号（**0-10mA**、**4-20mA**）、电流小信号（**mA**、 **μA**）。

开关量：**TTL**电平频率信号（**Hz**）、小频率信号、**TTL**电平开关信号、非**TTL**电平开关信号。

前向通道信号处理方式


模拟电压大信号（V）
A/D
单片机
模拟电压大信号（V）
V/F
单片机
模拟电压小信号（mV、μV）
放大
A/D
单片机
模拟电压小信号（mV、μV）
放大
V/F
单片机
电流大信号0~10mA、4~20mA
I/V
A/D
单片机
电流大信号0~10mA、4~20mA
I/V
V/F
单片机
电流小信号μA
I/V
放大
A/D
单片机
电流小信号μA
I/V
放大
V/F
单片机
TTL电平频率信号
单片机
小频率信号
放大
整形
单片机
TTL电平开关信号
单片机
非TTL电平开关信号
防抖
整形
单片机

A/D转换器及接口技术

A/D转换器（Analog To Digit Converter）：将模拟量转换为与之成比例的数字量的器件称为**A/D**转换器，常用**ADC**表示。

1、A/D转换器的性能指标

（1）分辨率：分辨率是指输出数字量变化一个相邻数码所需输入模拟电压的变化量。**A/D**转换器的分辨率定义为满刻度电压与2^n之比值，其中**n**为**ADC**的位数。

例：**12**位的**ADC**分辨率计算。

满刻度电压：为V_{REFH} - V_{REFL}，若V_{REFH} =5V，V_{REFL} =-5V，则满刻度电压为**10V**

ADC分辨率：**10V**×$(1/2)^{12}$ = **2.4mV**

ADC分辨率：**5V**×$(1/2)^{12}$ = **1.2mV**

ADC分辨率：**10V**×$(1/2)^{8}$ = **39.1mV**

A/D转换器的性能指标

- （2）转换速率与转换时间：转换速率是指A/D转换器每秒钟转换的次数转换时间是指完成一次A/D转换所需的时间（包括稳定时间）。转换时间是转换速率的倒数。
- （3）量化误差：有限分辨率A/D的阶梯状转移特性曲线与理想无限分辨率A/D的转移特性曲线（直线）之间的最大偏差称为量化误差。通常是1个或半个最小数字量的模拟变化量，表示为1LSB，1/2LSB。
- （4）线性度：实际A/D转换器的转移函数与理想直线的最大偏差。不包括量化误差、偏移误差（输入信号为零时，输出信号不为零的值）和满刻度误差（满度输出时，对应的输入信号与理想输入信号值之差）三种误差
- （5）量程：量程是指A/D能够转换的电压范围，如0～5V，-10～+10V等。
- （6）其他指标：内部/外部电压基准、失调（零点）温度系数、增益温度系数，以及电源电压变化抑制比等性能指标。

传感器

- 能够将非电信号转换成电模拟信号的敏感元件叫传感器。
- 按被测参量来分，传感器分为：

被测类别	被测参量
热工量	温度、热量、比热、压力、压差、真空度、流量、流速、风速
机械量	位移(线位移、角位移)，尺寸、形状，力、力矩、应力，重量、质量，转速、线速度，震动幅度、频率、加速度、噪声
物性和成分量	气体化学成分、液体化学成分，酸碱度(PH值)、咸度、浓度、粘度，密度、比重
状态量	颜色、透明度、磨损量、材料内部裂纹或缺陷、气体泄漏、表面质量

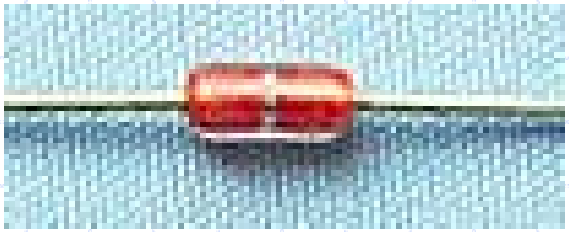
◆传感器

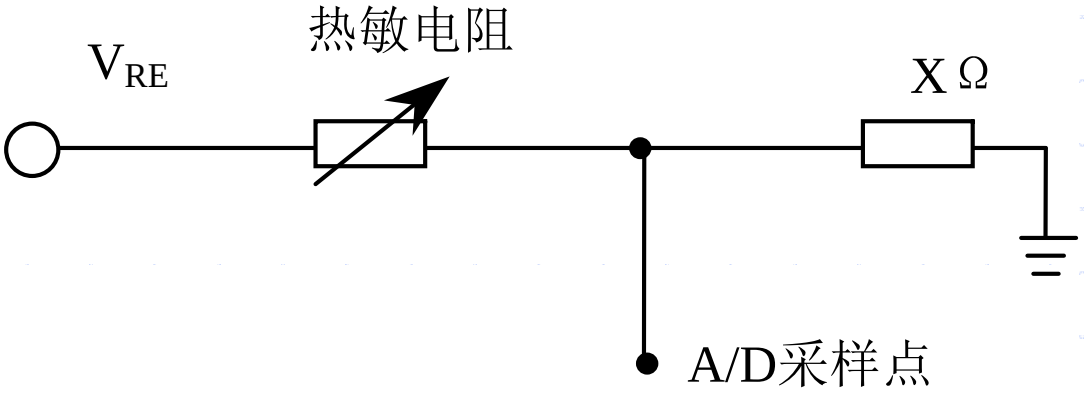
- 按被传感器的工作原理来分，传感器分为：

序号	工作原理	序号	工作原理
1	电阻式	8	光电式(包括红外式、光导纤维式)
2	电感式	9	谐振式
3	电容式	10	霍尔式(磁式)
4	阻抗式(电涡流式)	11	超声式
5	磁电式	12	同位素式
6	热电式	13	电化学式
7	压电式	14	微波式

常用传感器

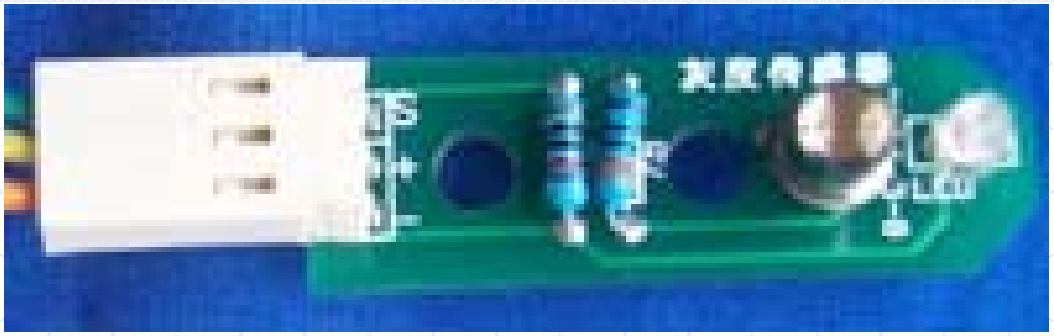
- 热敏电阻温度传感器





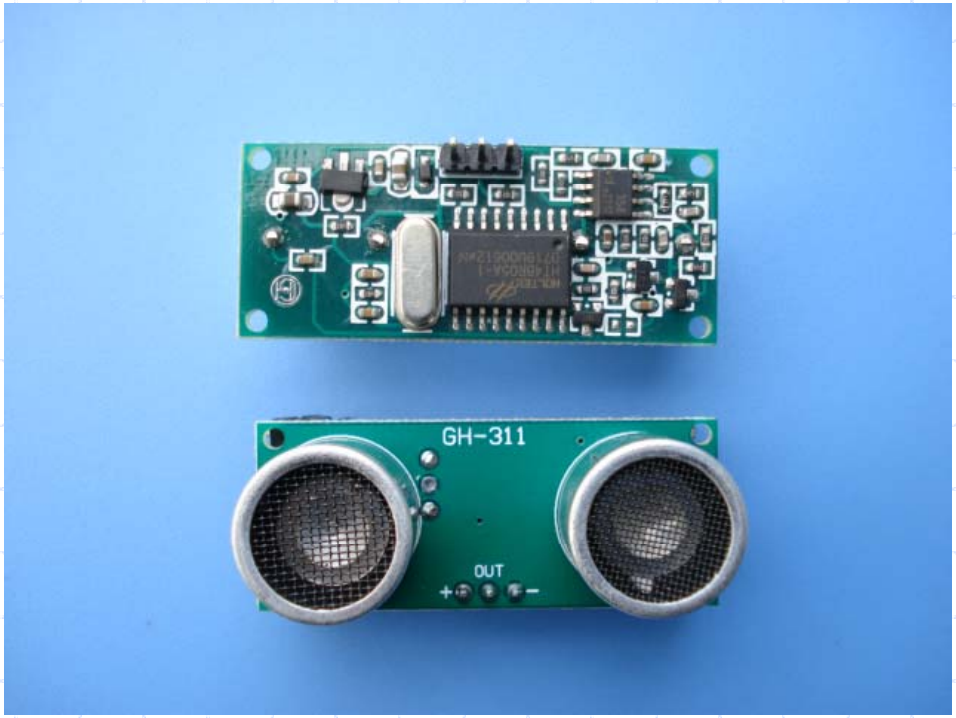
常用传感器

- 灰度传感器



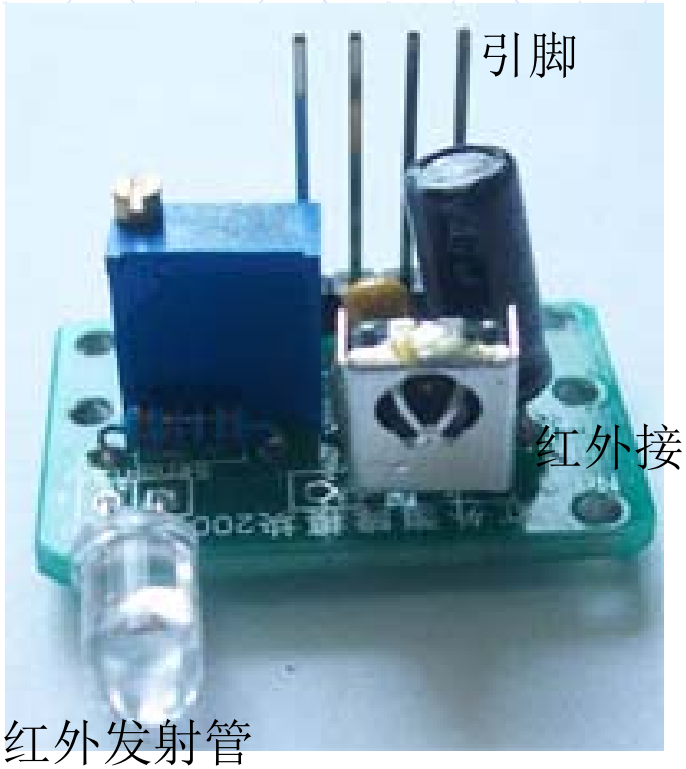
常用传感器

- 超声波传感器



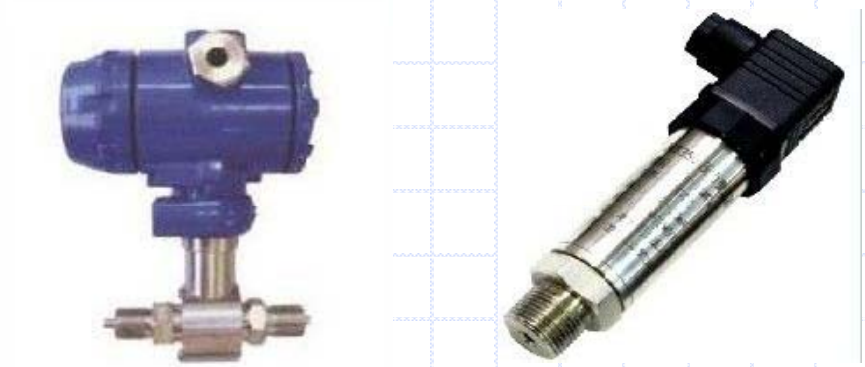
常用传感器

- 红外线传感器



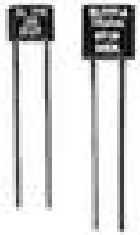
常用传感器

- 压力传感器

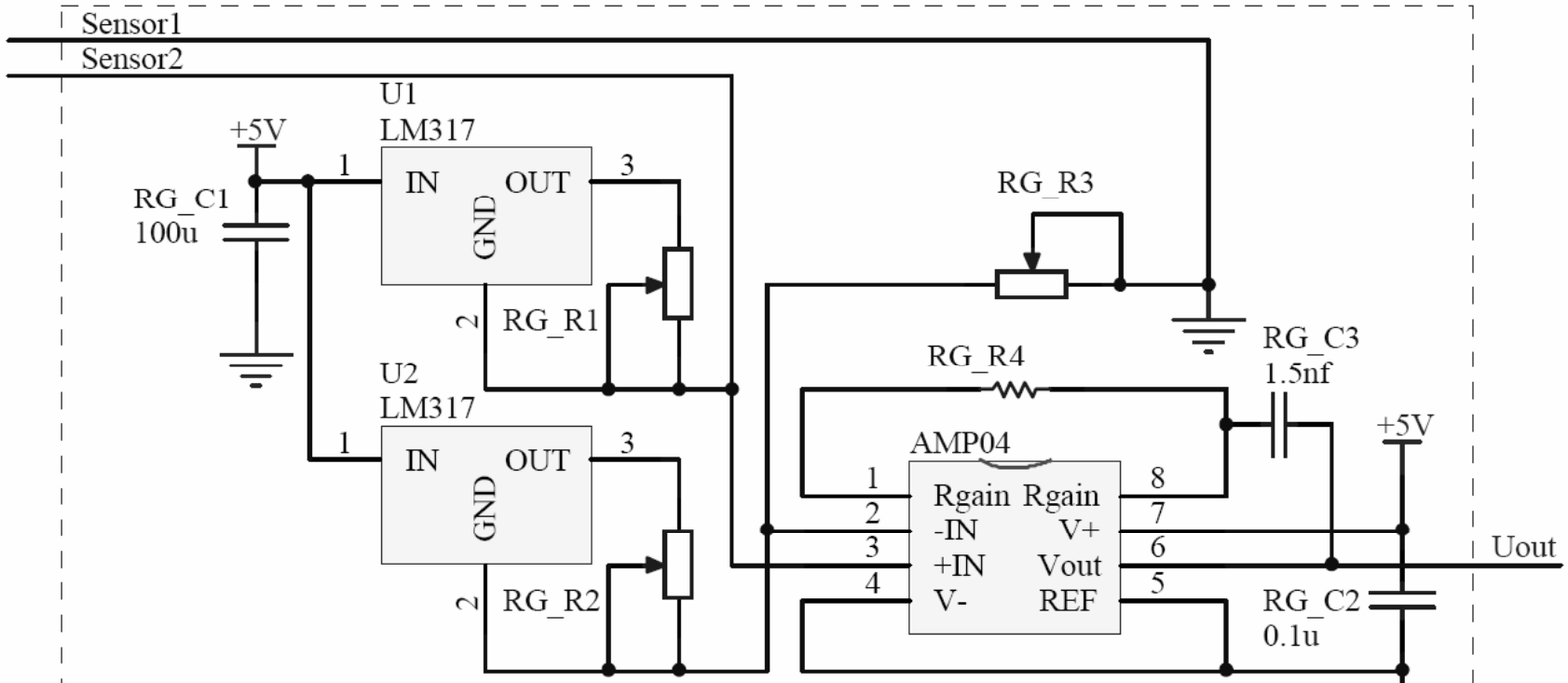


常用传感器

- 温度传感器



电阻型传感器采集电路由三部分组成：
传感器接口、恒流源电路和放大电路。



构件中文名称：电阻型传感器采集构件
构件英文名称：RG
构件使用说明：
（1）Sensor1和Sensor2分别接电阻型传感器两端，Uout为输出电压
（2）恒流源电流I=1.25V/RG_R1 (A)，范围10mA - 1.5A
（3）RG_R3为调零电阻，当RG_R3调节为传感器的最小电阻时，Uout=0(V)
（4）信号放大倍数Gain=100K/RG_R4

信号调理电路

- 传感器将被测信息转换成电信号输出，一般称为第一次变换。
- 第一次变换后的电信号具有以下特点：
  1、输出电信号一般较微弱，如μV~mV级或nA~mA级。
  2、输出电信号的信噪比较小，甚至有用信号淹没在噪声之中。
  3、传感器的输入/输出特性通常存在一定的非线性，并易受环境温度及周围电、磁干扰的影响。
  4、传感器的输出特性与电源的稳定性等有关，通常要求恒压或恒流供电。
- 所以，要对传感器信号进行信号调理，如：放大、滤波、线性化处理等。

几种常用的传感器

1. 温度传感器

温度传感器是通过感知被测对象温度的变化而相应改变某种特性或参量的敏感元件。

2. 光电传感器

光电传感器由能够完成光电转换的光电器件构成。

3. 光纤传感器

光纤传感器具有灵敏度高、电绝缘性能好、抗电磁干扰、耐腐蚀、耐高温、体积小、重量轻等优点，可广泛用于位移、速度、加速度、压力、温度、液位、流量、电流、磁场、放射性射线等物理量的测量。

4. 霍尔传感器

霍尔传感器以磁场为媒介，可以测量多种物理量。

5. CCD图像传感器

CCD图像传感器也称为电荷耦合图像传感器，它是由电荷耦合技术组成，集光电转换、图像存储、电荷转移为一体的固态图像传感器。

本讲课程：

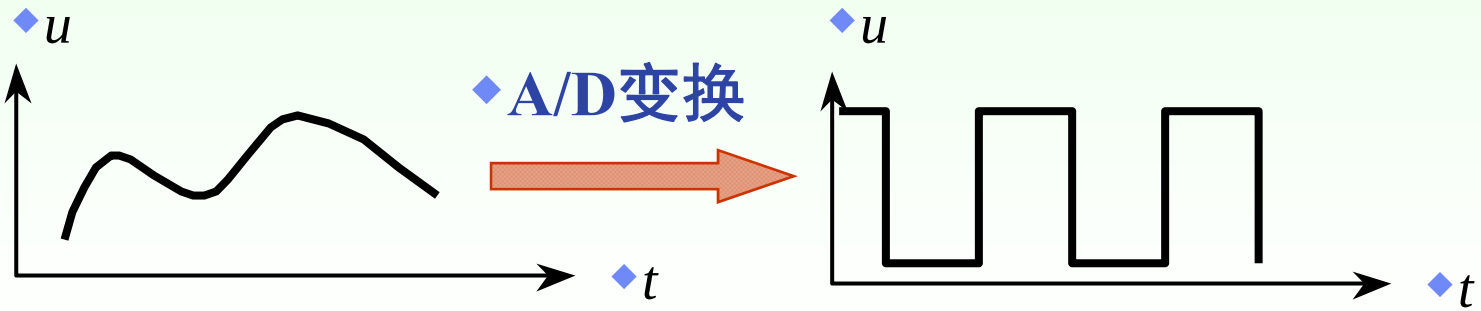
- 13.1 ARM硬件系统
- 13.2 信号的检测
- 13.3 A/D转换器原理
- 13.4 MCS51的A/D转换器
- 13.5 STM32的A/D转换器
- 13.6 D/A转换器应用

A/D转换器的基本原理

模拟信号和数字信号

模拟信号：在时间上和数值上连续的信号。

数字信号：在时间上和数值上不连续的（即离散的）信号。

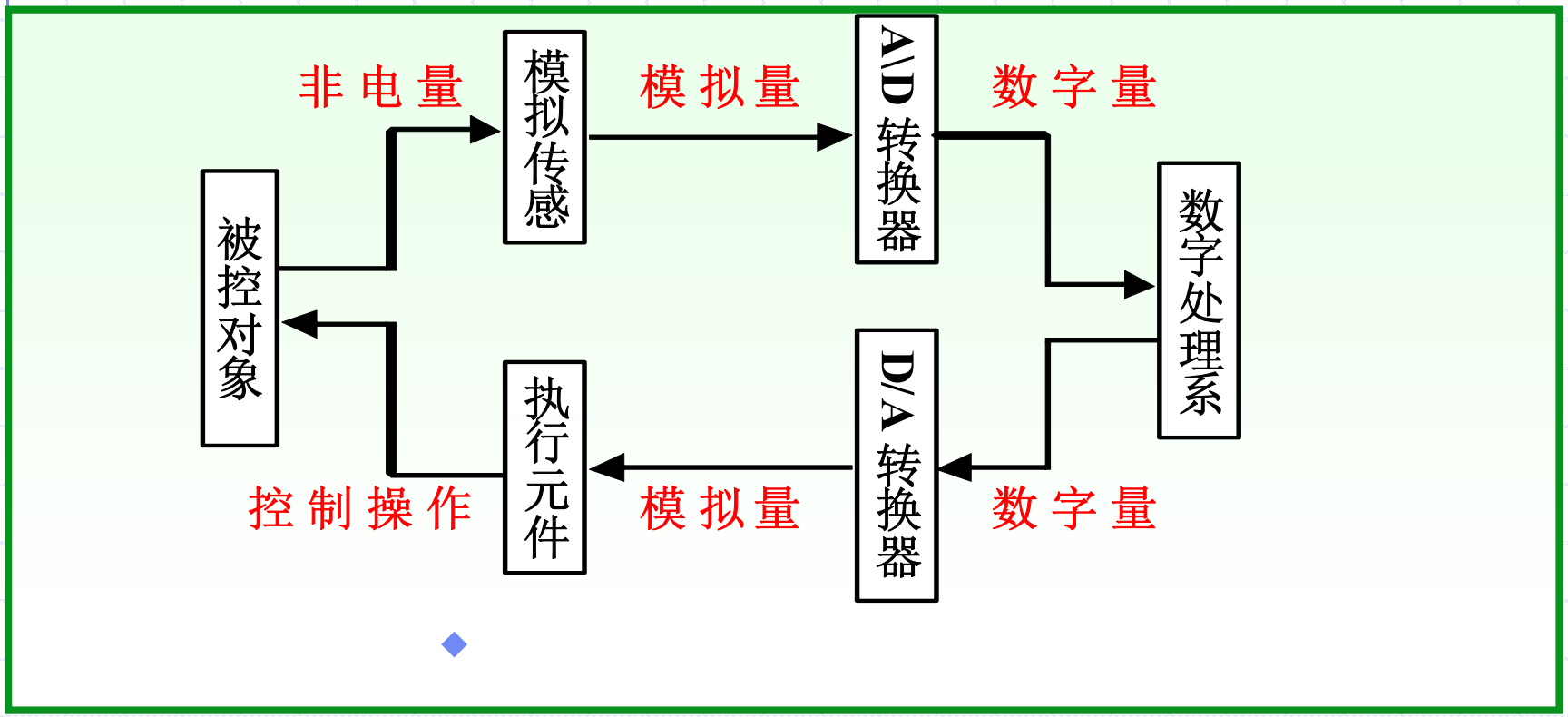


模拟信号波形　　数字信号波形

A/D转换器的基本原理

模数转换概述

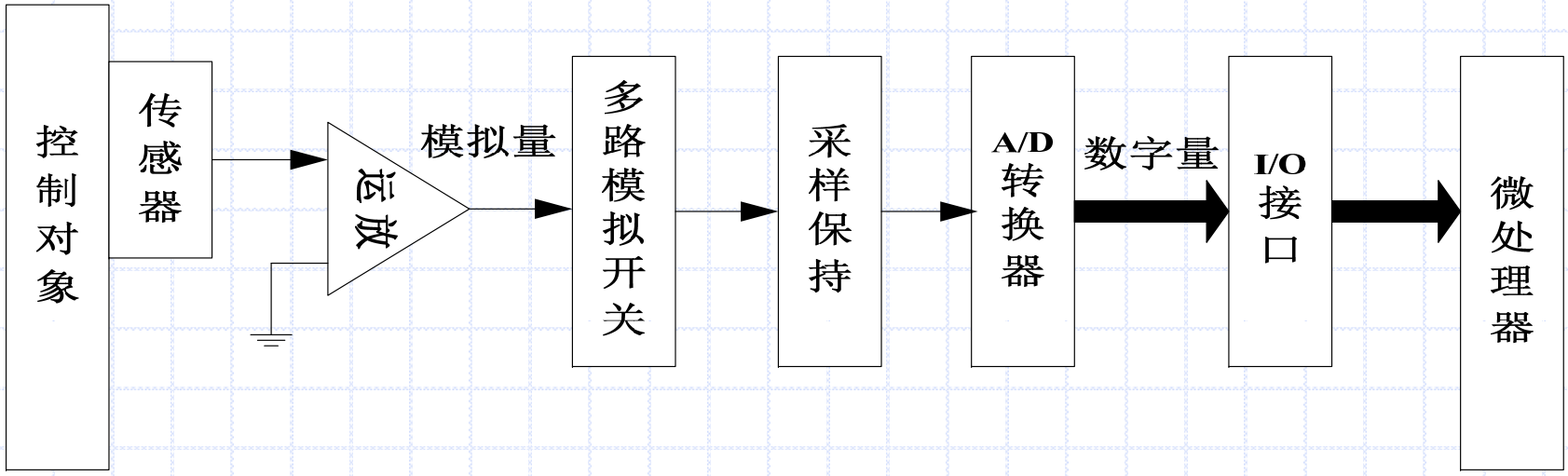
模数转换则是将模拟电量转换为数字量，使输出的数字量与输入的模拟电量成正比。实现这种转换功能的电路称为模数转换器（ADC）。



A/D(模/数)转换介绍

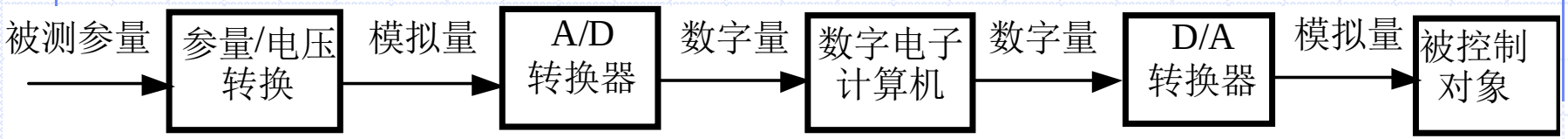
◆模拟信号的采集与处理

- 将现场模拟信号转变为数字信号并送入微处理器进行处理的系统称之为数据采集系统。数据采集系统由模拟信号采集、A/D转换、数字信号处理三大部分组成。



◆模拟信号采集示意图

A/D转换

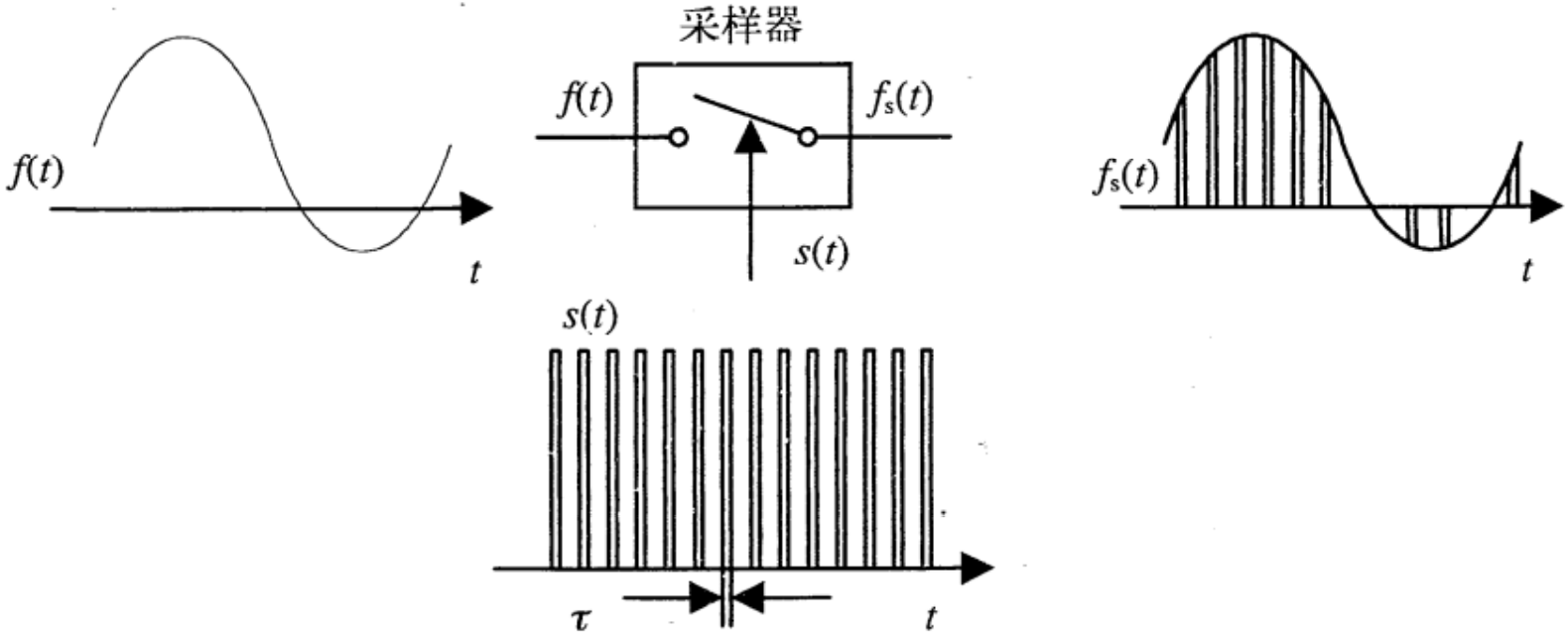


数字控制系统框图

1. 采样精度（分辨率）
2. 采样速率（转换速率）
3. 滤波
4. 物理量回归

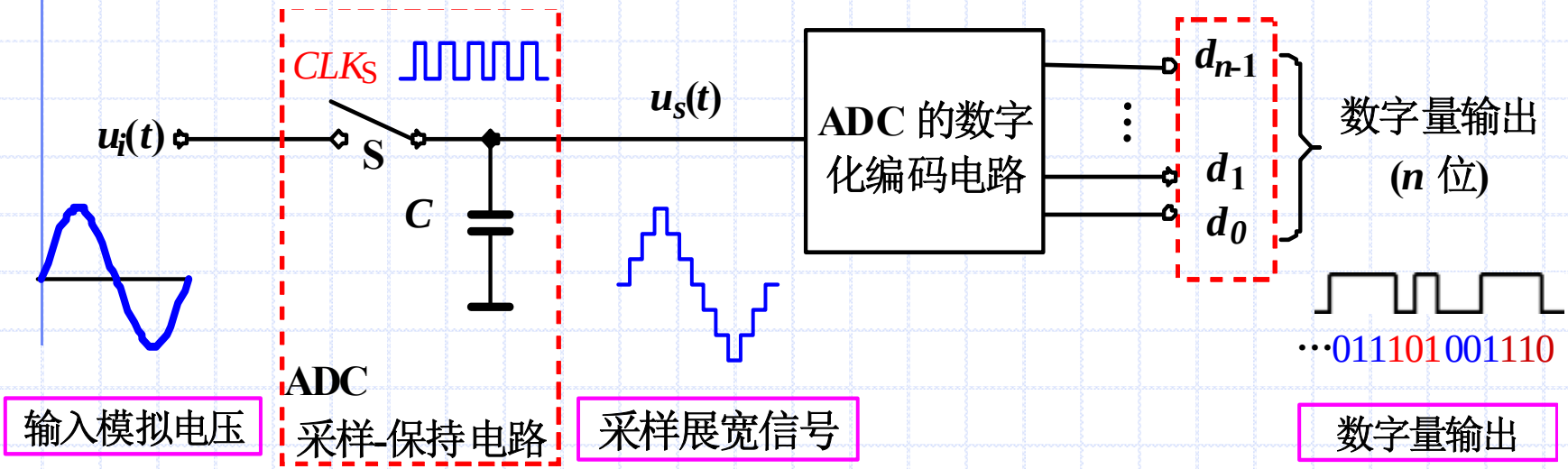
A/D(模/数)转换的原理--采样保持

- **A/D**转换又称模/数转换，就是把模拟信号数字化。**A/D**转换器首先要对模拟信号进行采样，然后量化成数字信号输出，**A/D**转换中要完成采样保持和量化编码两个内容。
  - 在连续变化的模拟量上按一定的规律取出其中某一些瞬时值(样点)来代表这个连续的模拟量。这个过程就是采样。



◆A/D转换器的基本原理

◆A/D转换器的基本原理



◆

◆显然，模数转换一般要分采样、保持、量化和编码四个步骤进行。

A/D转换器原理

- A/D转换的技术指标
  - 1、分辨率（Resolution）
    - 数字量变化一个最小量时模拟信号的变化量，$1/2^n$
  - 2、 转换速率（Conversion Rate）
    - 完成一次从模拟转换到数字的A/D转换所需的时间的倒数
  - 3、量化误差（Quantizing Error）
    - 由于A/D的有限分辨率而引起的误差（1LSB或1/2LSB）
  - 4、偏移误差（Offset Error）
    - 输入信号为零时输出信号不为零的值
  - 5、满刻度误差（Full Scale Error）
    - 满度输出时对应的输入信号与理想输入信号值之差
  - 6、线性度（Linearity）
    - 实际转换器的转移函数与理想直线的最大偏移

A/D(模/数)转换的原理--量化编码

- 量化编码：

  以一定的量化单位，把数值上连续的模拟量而时间上离散的模拟信号通过量化装置转变为数值上离散的阶跃量的过程。

- 常见的量化编码技术有：

  1、计数式转换；

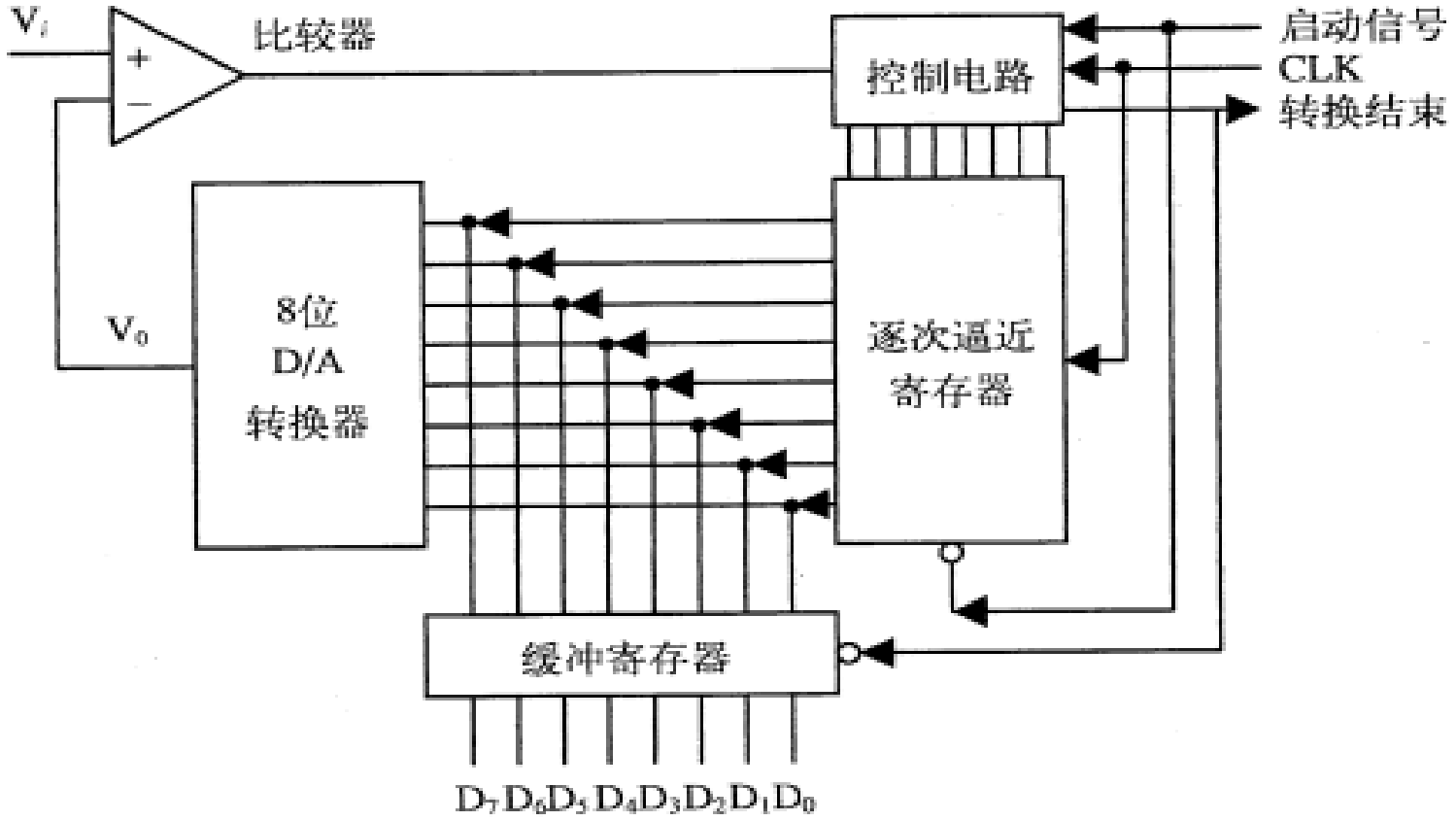
  2、双积分式转换；

  3、逐次逼近式转换；

  4、并联式转换。

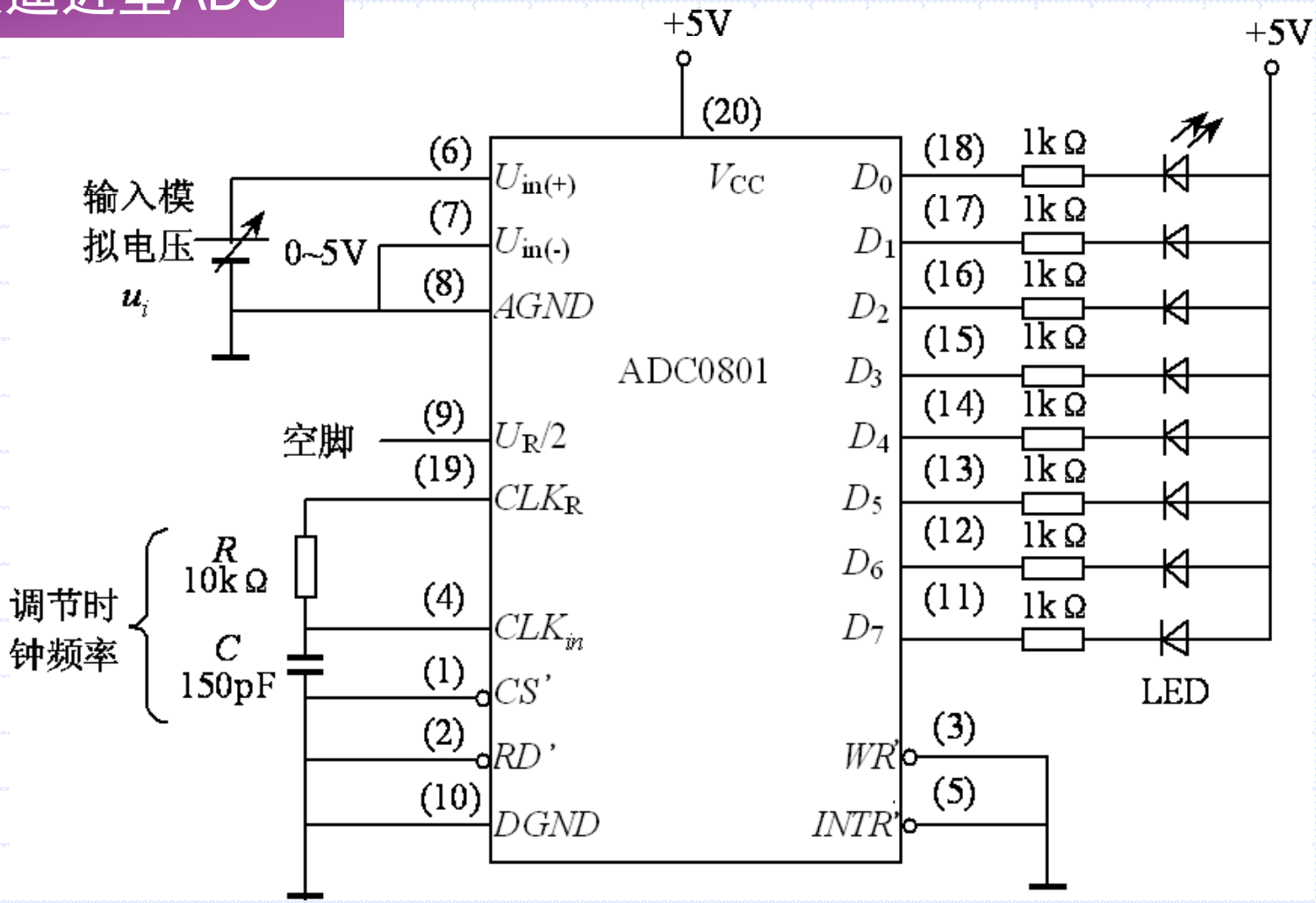
A/D(模/数)转换的原理--逐步逼近式转换

- 逐次逼近式A/D转换是使用最广泛的一种A/D转换方法，A/D转换集成电路芯片通常都采用这种方式工作。



◆A/D转换器的基本原理

◆逐次逼近型ADC



A/D(模/数)转换的原理--计数式转换

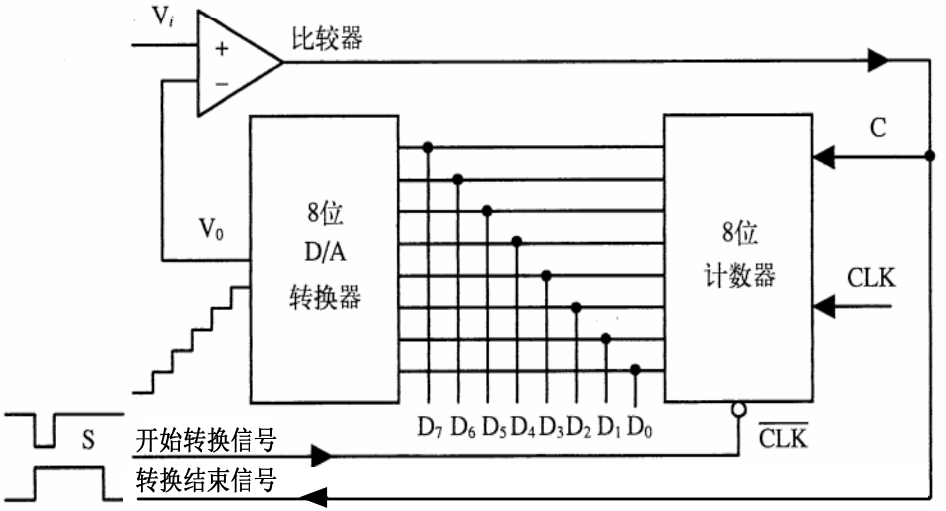
具体工作过程如下：

① 首先启动信号S由高电平变为低电平，使计数器的输出清0，当启动信号恢复高电平时，计数器准备计数。

② 转换器的输出电压为0时，运算放大器在同相端的输入电压作用下，使计数信号C =l。

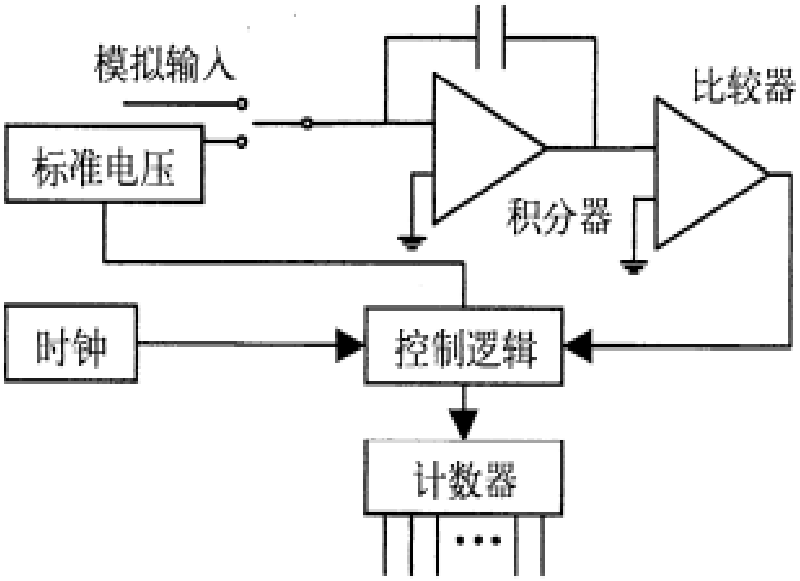
③ 计数器开始计数，在CLK时钟脉冲的驱动下，8位计数器输出的数字量不断增加，使得D/A转换器输出电压Vo不断上升。

④ 运算放大器的输出变为低电平时，计数器停止计数，这时候的数字输出量D7 ~ D0就是与模拟输入电压对应的数字量。

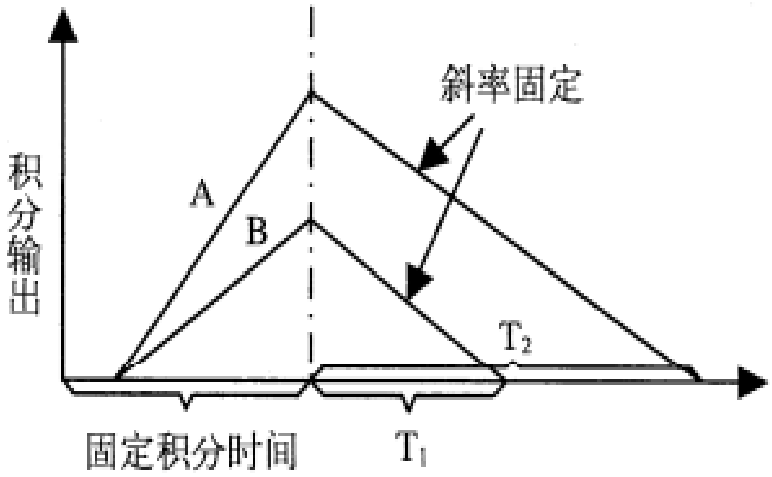


A/D(模/数)转换的原理--积分式转换

- 主要部件包括积分器、比较器、计数器和标准电压源。
- 用标准的高频时钟脉冲测定反向积分所用的时间，就可以得到输入模拟电压所对应的数字量，实现了A/D转换。
- 双积分式转换的优点是精度高、干扰小，但是速度较慢。



a) 电路工作原理



T_1和T_2正比于输入电压

b) 双积分原理

A/D转换器的基本原理

A/D转换器的主要技术指标

1、分辨率

A/D转换器的分辨率用输出二进制数的位数表示，位数越多，误差越小，转换精度越高。例如，输入模拟电压的变化范围为0～5V，输出8位二进制数可以分辨的最小模拟电压为$5V\times2^{-8}=20mV$；而输出12位二进制数可以分辨的最小模拟电压为$5V\times2^{-12}\approx1.22mV$。

2、相对精度

相对精度是指A/D转换器实际输出数字量与理论输出数字量之间的最大差值。通常用最低有效位LSB的倍数来表示。如相对精度不大于(1/2)LSB，就说明实际输出数字量与理论输出数字量的最大误差不超过(1/2)LSB。

3、转换速度

转换速度是指A/D转换器完成一次转换所需的时间。转换时间是指从接到转换控制信号开始，到输出端得到稳定的数字输出信号所经过的这段时间。

本讲课程：

- 13.1 ARM硬件系统
- 13.2 信号的检测
- 13.3 A/D转换器原理
- 13.4 MCS51的A/D转换器
- 13.5 STM32的A/D转换器
- 13.6 D/A转换器应用

ADC0809的应用

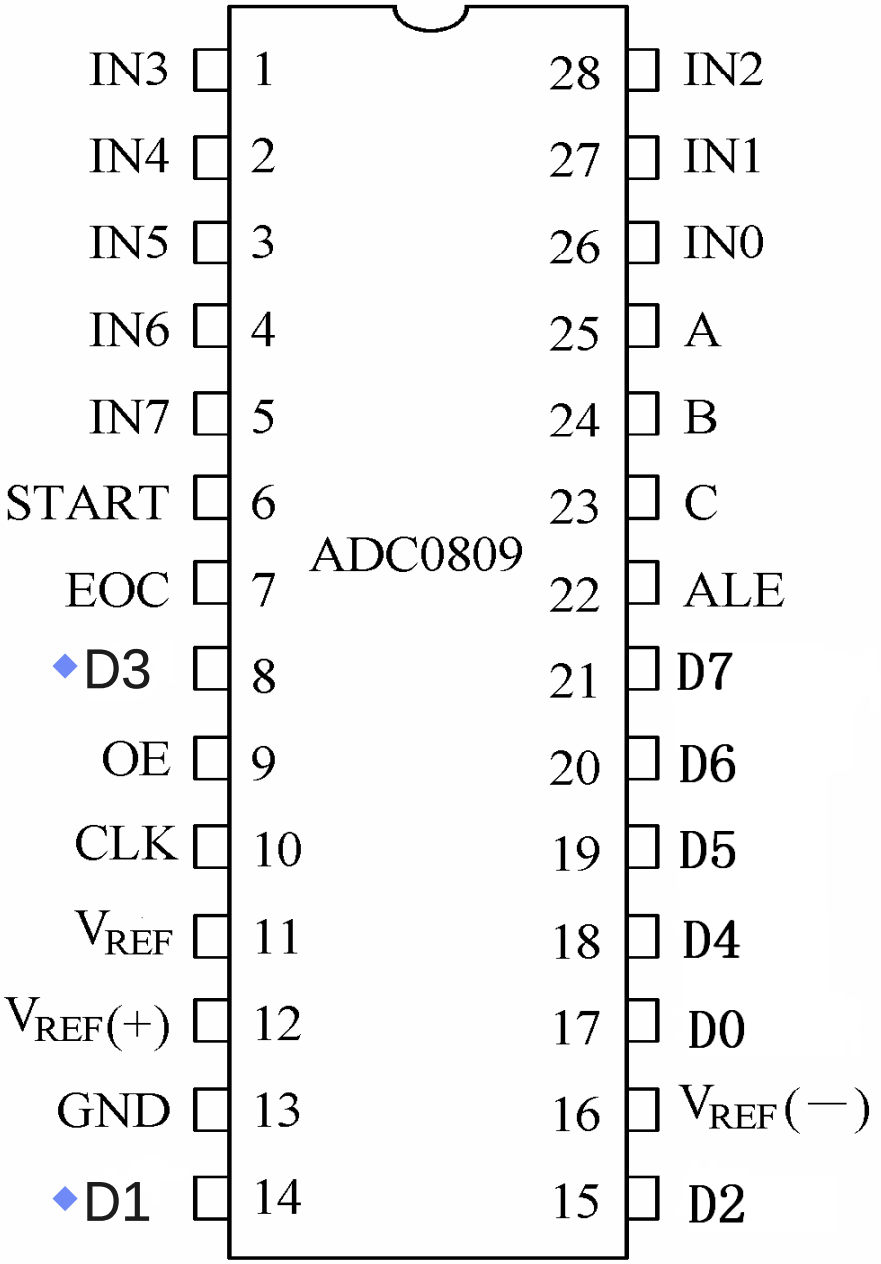
1、ADC0809的特点

ADC0809是NS（National Semiconductor，美国国家半导体）公司产品，主要性能指标：

- 分辨率为**8**位，误差**1LSB**
- 转换时间为**100μs**（外部时钟输入频率f_c **= 640 kHz** ）
- 单一电源**+5V**，采用单一电源**+5V**供电时量程为**0～5V**
- **8**路输入通道，带有锁存控制逻辑
- **DIP28**封装
- 转换结果读取方式有延时读数、查询**EOC=1**、**EOC**申请中断。

2、ADC0809引脚定义

- IN0—IN7：8通道模拟量输入端
- D0~D7 ： 8位数字量输出端
- C、B、A：地址信号，选择输入通道
- ALE： 地址锁存允许控制信号
- START：清0内寄存器,启动转换
- OE： 允许读A/D结果,高有效
- CLK：时钟输入端，范围为10kHz～1200kHz，典型值640kHz
- EOC：转换结束信号，高有效
- Vcc：+5V
- Vref+：参考电压，+5V
- Vref-：0V



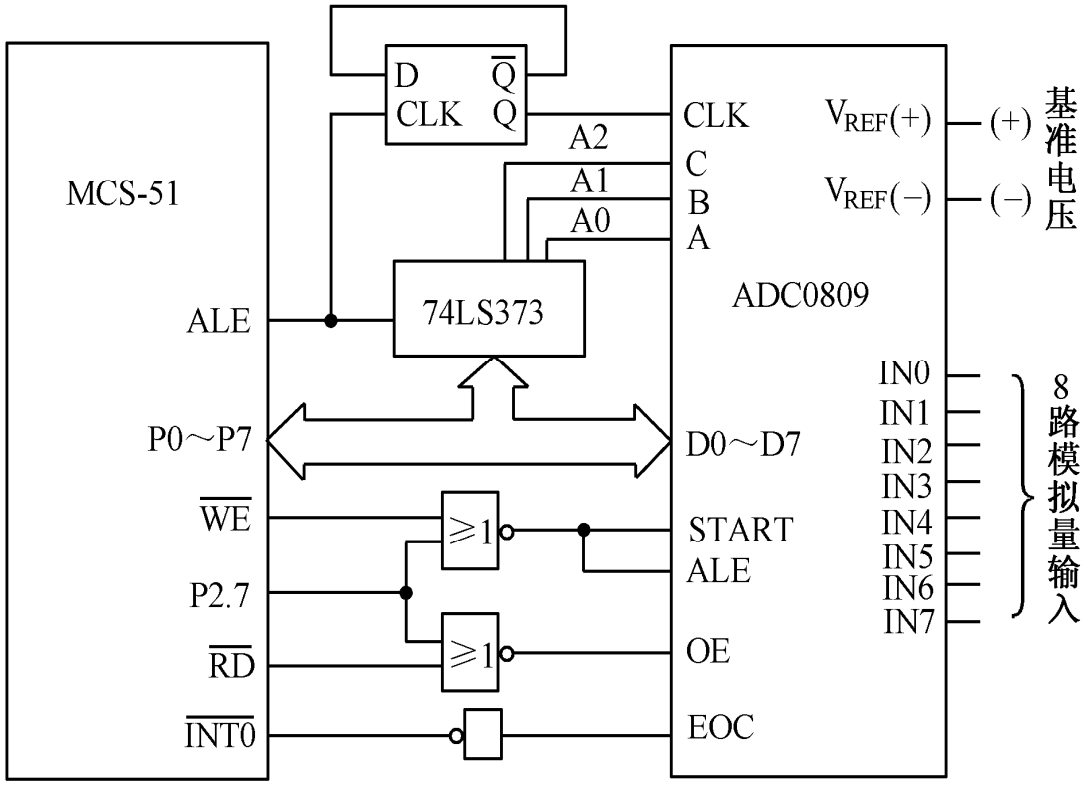
3、输入通道的选择

- ADC0809 有8路模拟量输入IN7～IN0 。一次只能选通其中的某一路进行转换，选通的通道由ALE上升沿时送入的C，B，A引脚信号决定。C，B，A地址与选通的通道间的关系如下表所示。

C　B　A	被选通的通道
0　0　0	IN0
0　0　1	IN1
0　1　0	IN2
0　1　1	IN3
1　0　0	IN4
1　0　1	IN5
1　1　0	IN6
1　1　1	IN7

4、ADC0809与单片机的接口单路

由于**ADC0809**数据输出可以直接连接**MCS-51**的数据总线**P0**口，可通过外部中断或查询方式读取**A/D**转换结果。



- **IN0地址：7FF8H**
- **IN1地址：7FF9H**
- **IN2地址：7FFAH**
- **IN3地址：7FFBH**
- **IN4地址：7FFCH**
- **IN5地址：7FFDH**
- **IN6地址：7FFEH**
- **IN7地址：7FFFH**

5、ADC0809转换程序设计

题目： 假设ADC0809与MCS-51的硬件连接如前页图所示，要求采用中断方法，进行8路A/D转换，将IN0～IN7转换结果分别存入片内RAM的30H～37H地址单元中。

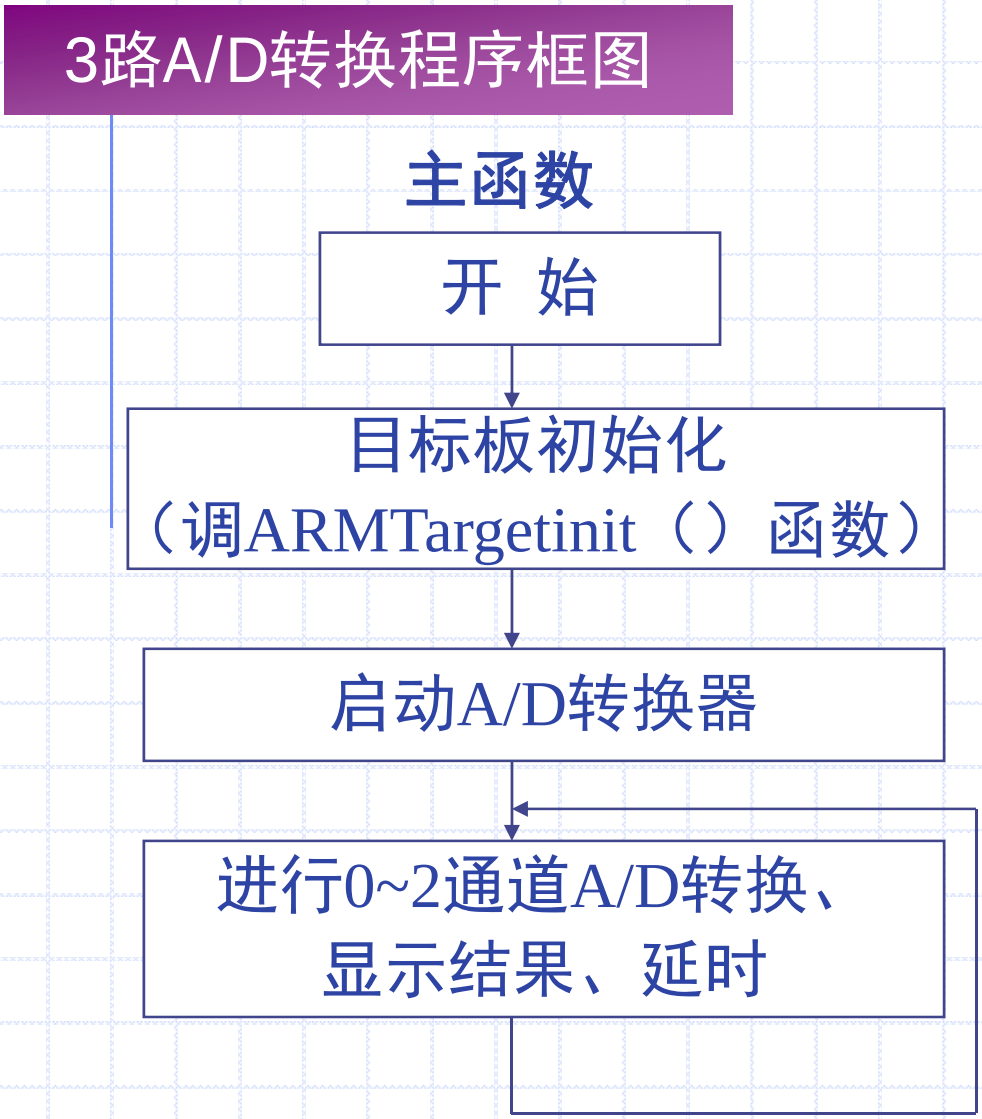
解：程序如下：

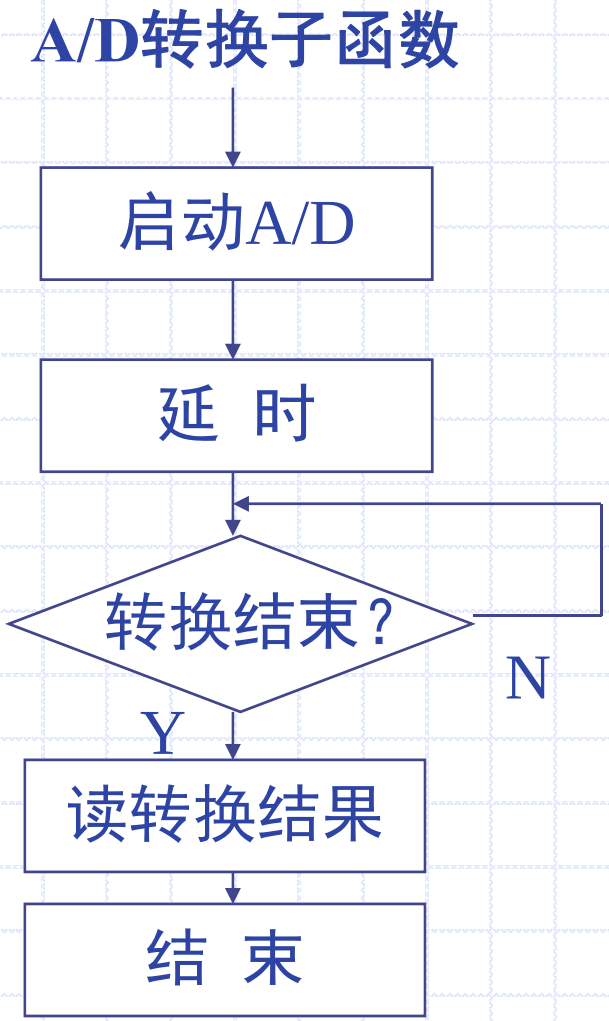
```
        ORG   0000H
        LJMP  MAIN          ;转主程序
        ORG   0003H         ;中断服务入口地址
        LJMP  INT0          ;中断服务。
        ORG   0100H
// 主程序
MAIN:   MOV   R0, #30H      ;内部数据指针指向30H单元
        MOV   DPTR, #7FF8H  ;指向P2.7口，且选通IN0
        SETB  IT0           ;设置下降沿触发
        SETB  EX0           ;允许中断
        SETB  EA            ;开总中断允许
        MOVX  @DPTR, A      ;启动A/D转换
        LJMP  $             ;等待转换结束中断
```

```
//中断服务程序
    INT0:  MOVX  A, @DPTR          ; 取A/D转换结果
           MOV  @R0, A             ; 存结果
           INC  R0                 ; 内部指针下移
           INC  DPTR               ; 外部指针下移，指向下一路
           CJNE  R0，#38H，NEXT    ; 未转换完8路，继续转换
           CLR  EX0                ; 关中断允许
           RETI                    ; 中断返回
    NEXT:  MOVX @DPTR, A           ; 启动下一路A/D转换
           RETI                    ; 中断返回，继续等待下一次
           END
```

A/D转换器编程

3路A/D转换程序框图





A/D转换器的操作

- 先将测量通道引脚设置为AINx功能，然后设置ADC的工作模式，ADC转换通道，转换时钟，并启动ADC转换。
- ADC的时钟频率不能大于4.5MHz。
- ADC转换时钟频率Fadclk的计算公式为：

  Fadclk = Fpclk/(CLKDIV+1)。

AIN0初始化示例

```
PINSEL1 = 0x00400000;          // 设置P0.27为AIN0功能
ADCR = (1<<0)                  // SEL=1，选择通道0
((Fpclk/1000000-1)<< 8)        // CLKDIV = Fpclk/1000000-1，即转换时钟为1MHz
(0<<16)                        // BURST=0，软件控制转换操作
(0<<17)                        // CLKS=0，使用11个时钟周期转换
(1<<21)                        // PDN=1，正常工作模式（非掉电模式）
(0<<22)                        // TEST[1:0]=00，正常工作模式（非测试模式）
(1<<24)                        // START=1，直接启动ADC转换
(0<<27);                       // EDGA=0
```

实验内容

- **实验内容1：**手动调节电位器的大小，利用A/D转换器对电位器输出的电压进行测量，A/D测量后的结果通过数码管显示。

- **实验内容2：** 对于外部的电压信号（范围为0~5V）进行测量，控制数据电位器，使数字电位器x9015输出引脚上的电平最大输出为3.3V，测量后的结果通过数码管显示。

A/D模块基本编程方法

- **1．A/D转换初始化**

  在程序初始化时应对A/D转换的5个控制寄存器写入控制字节，决定序列长度，设置分频系数和转换精度等。

- **2．启动A/D转换**

  对于A/D转换状态向控制寄存器写入控制字节，选取要转换的通道、结果寄存器的调整方式、设置是连续转换还是一次转换。

- **3．获得A/D转换结果**

  若是中断方式，在A/D中断程序中取得，若是查询方式，通过A/D转换状态寄存器0(ADC_SR)的第1位(EOC位)取得，当EOC=1时可从A/D数据寄存器中取的数据。

```
/*************************************************/
/*                  初始化AD模块                 */
/*************************************************/
void INIT_AD(void)
{
 ATD0CTL2 = 0x40;  //启动A/D模块,快速清零,禁止中断
 ATD0CTL1_SRES=0;  //选用8位模数转换
 ATD0CTL3 = 0x88;  //每次只转换一个通道
 ATD0CTL4 = 0x01;  //AD模块时钟
}
```

```
/*                    起动AD转换                    */
unsigned char AD_capture(unsigned char s)
{
 unsigned char AD_data;
 switch(s)
 {
  case 1:
    ATD0CTL5 = 0x01;          //转换AD01
    while(!ATD0STAT2_CCF0);
    AD_data = ATD0DR0L;
    break;
case 2:
    ATD0CTL5 = 0x00;          //转换AD00
    while(!ATD0STAT2_CCF0);
    AD_data = ATD0DR0L;
    break;
 }
 return(AD_data);
}
```

本讲课程：

- 13.1 ARM硬件系统
- 13.2 信号的检测
- 13.3 A/D转换器原理
- 13.4 MCS51的A/D转换器
- 13.5 STM32的A/D转换器
- 13.6 D/A转换器应用

STM32的AD转换器

- 1 ADC的硬件结构及特征
- 2 工作模式
- 3 ADC中断
- 4 ADC寄存器
- 5 ADC库函数
- 6 ADC程序设计

1. ADC的硬件结构及功能

- STM32F103有2个12位ADC（ADC1和ADC2），是逐次逼近型模拟/数字转换器。ADC的输入时钟不得超过14MHz，它是由PCLK2经分频产生。
- 它有多达18个通道，可测量16个外部和2个内部信号源。
- 各通道的A/D转换可以单次、连续、扫描或间断模式执行。
- ADC的结果可以左对齐或右对齐方式存储在16位数据寄存器中。

ADCx的输入通道

- ADC123_IN0->PA0
- ADC123_IN1->PA1
- ADC123_IN2->PA2
- ADC123_IN3->PA3
- ADC12_IN4->PA4
- ADC12_IN5->PA5
- ADC12_IN6->PA6
- ADC12_IN7->PA7
- ADC12_IN8->PB0
- **ADC12_IN9->PB1**
- **ADC123_IN10->PC0**
- **ADC123_IN11->PC1**
- **ADC123_IN12->PC2**
- **ADC123_IN13->PC3**
- **ADC12_IN14->PC4**
- **ADC12_IN15->PC5**

- **ADC3_IN4->PF6**
- **ADC3_IN5->PF7**
- **ADC3_IN6->PF8**
- **ADC3_IN7->PF9**
- **ADC3_IN8->PF10**

ADC主要特征

- **12**位分辨率
- 转换结束、注入转换结束和发生模拟看门狗事件时产生中断
- 单次和连续转换模式
- 从通道**0**到通道**n**的自动扫描模式
- 自校准
- 带内嵌数据一致性的数据对齐
- 采样间隔可以按通道分别编程
- 规则转换和注入转换均有外部触发选项
- 间断模式
- 双重模式(带**2**个或以上**ADC**的器件)
- **ADC**转换时间： **STM32F103xx**增强型产品：时钟为**56MHz**时为**1μs(**时钟为**72MHz**为**1.17μs)** 。
- **ADC**供电要求：**2.4V**到**3.6V**
- **ADC**输入范围：**VREF- ≤ VIN ≤ VREF+**
- 规则通道转换期间有**DMA**请求产生。

2. 工作模式

- 通道选择
- 单次转换模式
- 连续转换模式
- 扫描模式
- 间断模式
- 数据对齐
- 双**ADC**模式

通道选择

- 有**16**个多路通道。可以把转换组织成两组：规则组和注入组。在任意多个通道上以任意顺序进行的一系列转换构成成组转换。
- 规则组由多达**16**个转换组成。规则通道和它们的转换顺序在**ADC_SQRx**寄存器中选择。
- 注入组由多达**4**个转换组成。注入通道和它们的转换顺序在**ADC_JSQR**寄存器中选择。

单次转换模式

- 单次转换模式下，**ADC**只执行一次转换。
- 如果一个规则通道被转换：
  - 转换数据被储存在**16**位**ADC_DR**寄存器中
  - **EOC(**转换结束**)**标志被设置
  - 如果设置了**EOCIE**，则产生中断。
- 如果一个注入通道被转换：
  - 转换数据被储存在**16**位的**ADC_DRJ1**寄存器中
  - **JEOC(**注入转换结束**)**标志被设置
  - 如果设置了**JEOCIE**位，则产生中断。
- 然后**ADC**停止。

连续转换模式

- 在连续转换模式中，当前面**ADC**转换一结束马上就启动另一次转换。
- 如果一个规则通道被转换：
  - 转换数据被储存在**16**位的**ADC_DR**寄存器中
  - **EOC(**转换结束**)**标志被设置
  - 如果设置了**EOCIE**，则产生中断。
- 如果一个注入通道被转换：
  - 转换数据被储存在**16**位的**ADC_DRJ1**寄存器中
  - **JEOC(**注入转换结束**)**标志被设置
  - 如果设置了**JEOCIE**位，则产生中断。

扫描模式

- 此模式用来扫描一组模拟通道。
- **ADC**扫描所有被**ADC_SQRX**寄存器(对规则通道)或**ADC_JSQR**(对注入通道)选中的所有通道。
- 在每个组的每个通道上执行单次转换。在每个转换结束时，同一组的下一个通道被自动转换。
- 如果设置了寄存器**ADC_CR2**中的**CONT**位，转换不会在选择组的最后一个通道上停止，而是再次从选择组的第一个通道继续转换。

间断模式

- 规则组：此模式通过设置**ADC_CR1**寄存器上的**DISCEN**位激活。它可以用来执行一个短序列的**n**次转换**(n<=8)**，此转换是**ADC_SQRx**寄存器所选择的转换序列的一部分。
- 注入组：在一个外部触发事件后，该模式按通道顺序逐个转换**ADC_JSQR**寄存器中选择的序列。 一个外部触发信号可以启动**ADC_JSQR**寄存器选择的下一个通道序列的转换，直到序列中所有的转换完成为止。总的序列长度由**ADC_JSQR**寄存器的**JL[1:0]**位定义。

数据对齐

- ADC_CR2寄存器中的ALIGN位选择转换后数据储存的对齐方式。数据可以左对齐或右对齐。SEXT位是扩展的符号值。

◆**数据右对齐：**

注入组

SEXT	SEXT	SEXT	SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

规则组

0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

◆**数据左对齐：**

注入组

SEXT	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0

规则组

D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0

双ADC模式

- 在有2个或以上ADC模块的产品中，可以使用双ADC模式。
- 共有6种可能的模式：
  - — 同步注入模式
  - — 同步规则模式
  - — 快速交叉模式
  - — 慢速交叉模式
  - — 交替触发模式
  - — 独立模式

- 还有可以用下列方式组合使用上面的模式：
  - — 同步注入模式 + 同步规则模式
  - — 同步规则模式 + 交替触发模式
  - — 同步注入模式 + 交叉模式

3. ADC中断

- 规则和注入组转换结束时能产生中断，当模拟看门狗状态位被设置时也能产生中断。它们都有独立的中断使能位。
- **ADC1**和**ADC2**的中断映射在同一个中断向量上，而**ADC3**的中断有自己的中断向量。

- **ADC**中断

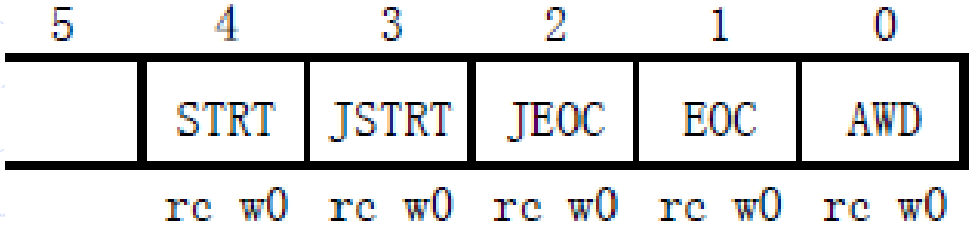
中断事件	事件标志	使能控制位
规则组转换结束	EOC	EOCIE
注入组转换结束	JEOC	JEOCIE
设置了模拟看门狗状态位	AWD	AWDIE

4. ADC寄存器

- SR --ADC状态寄存器
- CR1 --ADC控制寄存器1
- CR2 --ADC控制寄存器2
- SMPR1 --ADC采样时间寄存器1
- SMPR2 --ADC采样时间寄存器2
- JOFR1 --ADC注入通道偏移寄存器1
- JOFR2 --ADC注入通道偏移寄存器2
- JOFR3 --ADC注入通道偏移寄存器3
- JOFR4 --ADC注入通道偏移寄存器4

- HTR --ADC看门狗高阈值寄存器
- LTR --ADC看门狗低阈值寄存器
- SQR1 --ADC规则序列寄存器1
- SQR2 --ADC规则序列寄存器2
- SQR3 --ADC规则序列寄存器3
- JSQR1 --ADC注入序列寄存器
- JDR1 --ADC注入数据寄存器1
- JDR2 --ADC注入数据寄存器2
- JDR3 --ADC注入数据寄存器3
- JDR4 --ADC注入数据寄存器4
- DR --规则数据寄存器

ADC状态寄存器(ADC_SR)



- **STRT**： 规则通道开始位 **(Regular channel Start flag)**
- **JSTRT**：注入通道开始位 **(Injected channel Start flag)**
- **JEOC**：注入通道转换结束位 **(Injected channel end of conversion)**
- **EOC**：转换结束位 **(End of conversion)**
- **AWD**：模拟看门狗标志位 **(Analog watchdog flag)**。 该位由硬件在转换的电压值超出了**ADC_LTR**和**ADC_HTR**寄存器定义的范围时设置，由软件清除。

ADC控制寄存器1(ADC_CR1)

- **AWDEN**：在规则通道上开启模拟看门狗
- **JAWDEN**：在注入通道上开启模拟看门狗
- **DUALMOD[3:0]**：双模式选择 **(Dual mode selection)**， 软件使用这些位选择操作模式。
- **DISCNUM[2:0]**：间断模式通道计数
- **JDISCEN**：在注入通道上的间断模式
- **DISCEN**：在规则通道上的间断模式
- **JAUTO**：自动的注入通道组转换 **(Automatic Injected Group conversion)**
- **AWDSGL**：扫描模式中在一个单一的通道上使用看门狗 。
- **SCAN**：扫描模式 **(Scan mode)**
- **JEOCIE**：允许产生注入通道转换结束中断
- **AWDIE**：允许产生模拟看门狗中断
- **EOCIE**：允许产生**EOC**中断
- **AWDCH[4:0]**：模拟看门狗通道选择位

ADC控制寄存器2(ADC_CR2)

- **TSVREFE**：温度传感器和**VREFINT**使能。
- **SWSTART**：开始转换规则通道 **(Start conversion of regular channels)**
- **JSWSTART**：开始转换注入通道 **(Start conversion of injected channels)**
- **EXTTRIG**：规则通道的外部触发转换模式 。
- **EXTSEL[2:0]**：选择启动规则通道组转换的外部事件
- **JEXTTRIG**：注入通道的外部触发转换模式
- **JEXTSEL[2:0]**：选择启动注入通道组转换的外部事件
- **ALIGN**：数据对齐 **(Data alignment)**。 0：右对齐； 1：左对齐。
- **DMA**：直接存储器访问模式 **(Direct memory access mode)**
- **RSTCAL**：复位校准 **(Reset calibration)**
- **CAL**：**A/D**校准 **(A/D Calibration)**
- **CONT**：连续转换 **(Continuous conversion)**。 0：单次转换；

EXTSEL[2:0]，启动规则通道组转换的外部事件

ADC1和ADC2的触发配置如下

000：定时器1的CC1事件

100：定时器3的TRGO事件

001：定时器1的CC2事件

101：定时器4的CC4事件

010：定时器1的CC3事件

110：EXTI线11/ TIM8_TRGO事件，仅大容量产品具有该功能

011：定时器2的CC2事件

111：SWSTART

ADC采样时间寄存器1(ADC_SMPR1)

- **SMPx[2:0]：选择通道x的采样时间 (Channel x Sample time selection)**
- 这些位用于独立地选择每个通道的采样时间。

  000：1.5周期

  001：7.5周期

  010：13.5周期

  011：28.5周期

  100：41.5周期

  101：55.5周期

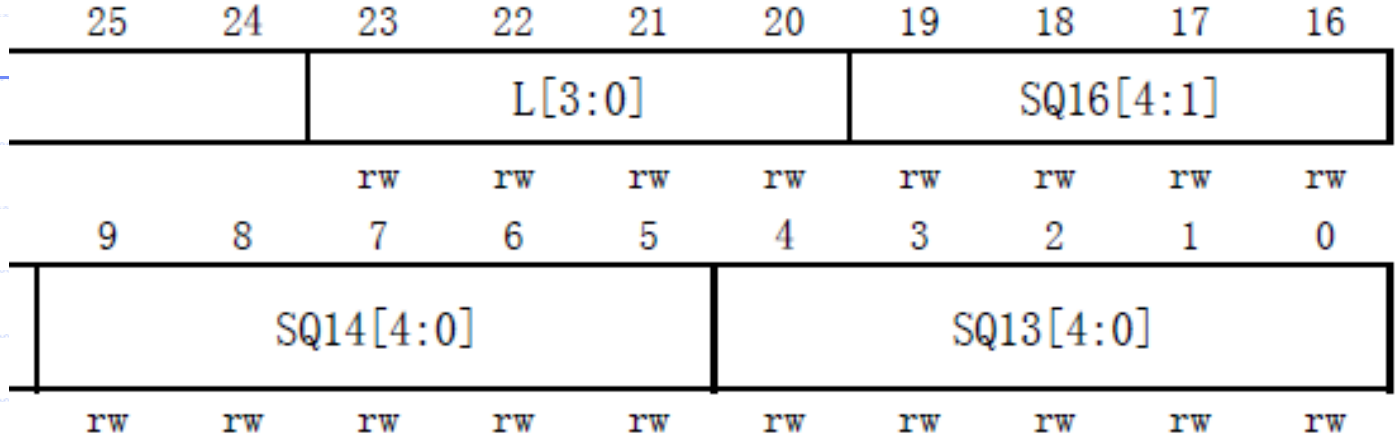
  110：71.5周期

  111：239.5周期

ADC注入通道数据偏移寄存器x (ADC_JOFRx)(x=1..4)

- JOFFSETx[11:0]：注入通道x的数据偏移 (Data offset for injected channel x) 当转换注入通道时，这些位定义了用于从原始转换数据中减去的数值。
- 转换的结果可以在ADC_JDRx寄存器中读出。

ADC规则序列寄存器1(ADC_SQR1)



- **L[3:0]**：规则通道序列长度。
- **SQ16[4:0]**：规则序列中的第**16**个转换，这些位由软件定义转换序列中的第16个转换通道的编号(0~17)。
- **SQ15[4:0]**：规则序列中的第**15**个转换
- 。。。
- **SQ13[4:0]**：规则序列中的第**13**个转换 **(13th conversion in regular sequence)**

L[3:0]：规则通道序列长度 (Regular channel sequence length)
这些位由软件定义在规则通道转换序列中的通道数目。
0000：1个转换
0001：2个转换
.....
1111：16个转换

SQ16[4:0]：规则序列中的第16个转换 (16th conversion in regular sequence)
这些位由软件定义转换序列中的第16个转换通道的编号(0~17)。

SQ15[4:0]：规则序列中的第15个转换 (15th conversion in regular sequence)

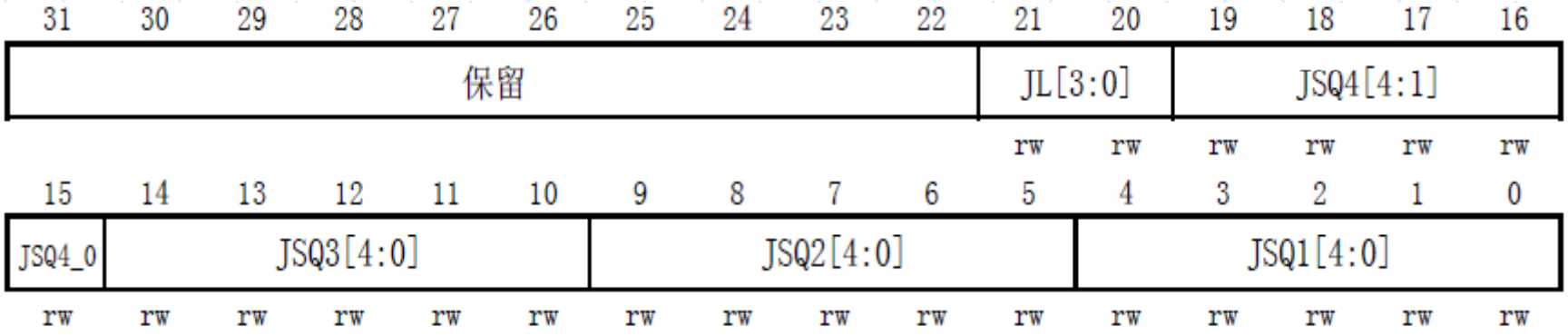
SQ14[4:0]：规则序列中的第14个转换 (14th conversion in regular sequence)

SQ13[4:0]：规则序列中的第13个转换 (13th conversion in regular sequence)

ADC注入序列寄存器(ADC_JSQR)

- **JL[1:0]**：注入通道序列长度 **(Injected sequence length)**

  (00：1个转换；01：2个转换；10：3个转换；11：4个转换)
- **JSQ4[4:0]**：注入序列中的第**4**个转换 。**(4th conversion in injected sequence** ）
- **JSQ3[4:0]**：注入序列中的第**3**个转换 **(3rd conversion in injected sequence)**



ADC 注入数据寄存器x (ADC_JDRx) (x= 1..4)

- **JDATA[15:0]**：注入转换的数据 **(Injected data)** 这些位为只读，包含了注入通道的转换结果。数据是左对齐或右对齐

- ADC规则数据寄存器(ADC_DR)
  - **ADC2DATA[15:0]**：**ADC2转换的数据 (ADC2 data)**
  - **DATA[15:0]**：规则转换的数据 **(Regular data)**

5. ADC库函数

- **ADC_DeInit**： 将外设**ADCx**的全部寄存器重设为缺省值
- **ADC_Init**： 根据**ADC_InitStruct**中指定的参数初始化外设**ADCx**的寄存器
- **ADC_StructInit**： 把**ADC_InitStruct**中的每一个参数按缺省值填入
- **ADC_Cmd**： 使能或者失能指定的**ADC**
- **ADC_DMACmd**： 使能或者失能指定的**ADC**的**DMA**请求
- **ADC_ITConfig**： 使能或者失能指定的**ADC**的中断
- **ADC_SoftwareStartConvCmd** 使能或者失能指定的**ADC**的软件转换启动功能
- **ADC_GetConversionValue**： 返回最近一次**ADCx**规则组的转换结果
- **ADC_RegularChannelConfig** 设置指定**ADC**的规则组通道，设置它们的转化顺序和采样时间
- **ADC_GetFlagStatus** ： 检查制定**ADC**标志位置**1**与否
- **ADC_ClearFlag**： 清除**ADCx**的待处理标志位
- **ADC_GetITStatus** ： 检查指定的**ADC**中断是否发生
- **ADC_ClearITPendingBit** ： 清除**ADCx**的中断待处理位

函数ADC_DeInit

- 功能描述: 将外设ADCx的全部寄存器重设为缺省值
- 函数原形:void ADC_DeInit(ADC_TypeDef* ADCx)

- 例：
```
/* Resets ADC2 */
ADC_DeInit(ADC2);
```

函数**ADC_Init**

- 功能描述: 根据**ADC_InitStruct**中指定的参数初始化外设**ADCx**的寄存器
- 函数原形:

void ADC_Init (ADC_TypeDef* ADCx,
ADC_InitTypeDef* ADC_InitStruct)

ADC_InitTypeDef structure

ADC_InitTypeDef定义于文件“stm32f10x_adc.h”：

```
 typedef struct
{
u32  ADC_Mode;
FunctionalState ADC_ScanConvMode;
FunctionalState ADC_ContinuousConvMode;
u32 ADC_ExternalTrigConv;
u32 ADC_DataAlign;
u8   ADC_NbrOfChannel;
} ADC_InitTypeDef
```

ADC_Mode:

ADC_Mode	描述
ADC_Mode_Independent	ADC1 和 ADC2 工作在独立模式
ADC_Mode_RegInjecSimult	ADC1 和 ADC2 工作在同步规则和同步注入模式
ADC_Mode_RegSimult_AlterTrig	ADC1 和 ADC2 工作在同步规则模式和交替触发模式
ADC_Mode_InjecSimult_FastInterl	ADC1 和 ADC2 工作在同步规则模式和快速交替模式
ADC_Mode_InjecSimult_SlowInterl	ADC1 和 ADC2 工作在同步注入模式和慢速交替模式
ADC_Mode_InjecSimult	ADC1 和 ADC2 工作在同步注入模式
ADC_Mode_RegSimult	ADC1 和 ADC2 工作在同步规则模式
ADC_Mode_FastInterl	ADC1 和 ADC2 工作在快速交替模式
ADC_Mode_SlowInterl	ADC1 和 ADC2 工作在慢速交替模式
ADC_Mode_AlterTrig	ADC1 和 ADC2 工作在交替触发模式

ADC_ScanConvMode

- **ADC_ScanConvMode**

  ADC_ScanConvMode规定了模数转换工作在扫描模式（多通道）还是单次（单通道）模式。可以设置这个参数为**ENABLE**或者**DISABLE**。

- **ADC_ContinuousConvMode**

  ADC_ContinuousConvMode规定了模数转换工作在连续还是单次模式。可以设置这个参数为**ENABLE**或者**DISABLE**。

- **ADC_ExternalTrigConv**

  ADC_ExternalTrigConv定义了使用外部触发来启动规则通道的模数转换，

ADC_ExternalTrigConv定义表

ADC_ExternalTrigConv	描述
ADC_ExternalTrigConv_T1_CC1	选择定时器 1 的捕获比较 1 作为转换外部触发
ADC_ExternalTrigConv_T1_CC2	选择定时器 1 的捕获比较 2 作为转换外部触发
ADC_ExternalTrigConv_T1_CC3	选择定时器 1 的捕获比较 3 作为转换外部触发
ADC_ExternalTrigConv_T2_CC2	选择定时器 2 的捕获比较 2 作为转换外部触发
ADC_ExternalTrigConv_T3_TRGO	选择定时器 3 的 TRGO 作为转换外部触发
ADC_ExternalTrigConv_T4_CC4	选择定时器 4 的捕获比较 4 作为转换外部触发
ADC_ExternalTrigConv_Ext_IT11	选择外部中断线 11 事件作为转换外部触发
ADC_ExternalTrigConv_None	转换由软件而不是外部触发启动

ADC_DataAlign

- 规定了ADC数据向左边对齐还是向右边对齐。

ADC_DataAlign	描述
ADC_DataAlign_Right	ADC 数据右对齐
ADC_DataAlign_Left	ADC 数据左对齐

ADC_NbrOfChannel

- 规定了顺序进行规则转换的**ADC**通道的数目。这个数目的取值范围是**1**到**16**。

例： /* Initialize the ADC1 according to the ADC_InitStructure members */

```
ADC_InitTypeDef ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv =
    ADC_ExternalTrigConv_Ext_IT11;
ADC_InitStructure.ADC_DataAlign =
    ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 16;
ADC_Init(ADC1, &ADC_InitStructure);
```

函数ADC_Cmd

- 功能描述: 使能或者失能指定的ADC
- 例： /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);
- 注意：函数ADC_Cmd只能在其他ADC设置函数之后被调用。

函数ADC_ResetCalibration

- 功能描述:重置指定的ADC的校准寄存器
- 例： /* Reset the ADC1 Calibration registers */
  ADC_ResetCalibration(ADC1);

函数ADC_GetResetCalibrationStatus

- 功能描述：获取**ADC**重置校准寄存器的状态
- 返回值 ：**ADC**重置校准寄存器的新状态（**SET**或者**RESET**）

- 例： /* Get the ADC2 reset calibration registers status */

```
FlagStatus Status;
Status = ADC_GetResetCalibrationStatus(ADC2);
```

函数**ADC_StartCalibration**

- 功能描述： 开始指定**ADC**的校准状态
- 例： /* Start the ADC2 Calibration */
  ADC_StartCalibration(ADC2);

函数ADC_GetCalibrationStatus

- 功能描述： 获取指定**ADC**的校准程序

- 例： /* Get the ADC2 calibration status */
  FlagStatus Status;
  Status = ADC_GetCalibrationStatus(ADC2);

函数ADC_SoftwareStartConvCmd

- 功能描述：使能或者失能指定的**ADC**的软件转换启动功能

- 例：
```
/* Start by software the ADC1 Conversion */
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
```

函数RCC_ADCCLKConfig

- 功能描述：设置**ADC**时钟（**ADCCLK**）
- 函数原形：

void ADC_ADCCLKConfig(u32 RCC_ADCCLKSource)

- 例： /* Configure ADCCLK such as ADCCLK = PCLK2/2 */
  RCC_ADCCLKConfig(RCC_PCLK2_Div2);

RCC_ADCCLKSource	描述
RCC_PCLK2_Div2	ADC 时钟 = PCLK / 2
RCC_PCLK2_Div4	ADC 时钟 = PCLK / 4
RCC_PCLK2_Div6	ADC 时钟 = PCLK / 6
RCC_PCLK2_Div8	ADC 时钟 = PCLK / 8

函数**ADC_RegularChannelConfig**

- 功能描述：设置指定**ADC**的规则组通道，设置它们的转化顺序和采样时间
- 函数原形**void** ADC_RegularChannelConfig(ADC_TypeDef* ADCx,

  u8 ADC_Channel，u8 Rank，u8 ADC_SampleTime)
- **ADCx**：**x**可以是**1**或者**2**来选择**ADC**外设**ADC1**或**ADC2**
- **ADC_Channel**：被设置的**ADC**通道
- **Rank**：规则组采样顺序。取值范围**1**到**16**
- **ADC_SampleTime**：指定**ADC**通道的采样时间值

ADC_Channel值

ADC_Channel_0：　选择ADC通道0

ADC_Channel_1 ：　选择ADC通道1

ADC_Channel_17：　选择ADC通道17

ADC_SampleTime值：

ADC_SampleTime	描述
ADC_SampleTime_1Cycles5	采样时间为 1.5 周期
ADC_SampleTime_7Cycles5	采样时间为 7.5 周期
ADC_SampleTime_13Cycles5	采样时间为 13.5 周期
ADC_SampleTime_28Cycles5	采样时间为 28.5 周期
ADC_SampleTime_41Cycles5	采样时间为 41.5 周期
ADC_SampleTime_55Cycles5	采样时间为 55.5 周期
ADC_SampleTime_71Cycles5	采样时间为 71.5 周期
ADC_SampleTime_239Cycles5	采样时间为 239.5 周期

- 例： /* Configures ADC1 Channel2 as: first converted channel with an 7.5 cycles sample time */

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 1,
    ADC_SampleTime_7Cycles5);
```

- /* Configures ADC1 Channel8 as: second converted channel with an 1.5 cycles sample time */

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 2,
    ADC_SampleTime_1Cycles5);
```

函数**ADC_GetFlagStatus**

- 功能描述： 检查制定**ADC**标志位置**1**与否
- 例： /* Test if the ADC1 EOC flag is set or not */

```
FlagStatus Status;
Status = ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC);
```

Table 49. ADC_FLAG 的值

ADC_AnalogWatchdog	描述
ADC_FLAG_AWD	模拟看门狗标志位
ADC_FLAG_EOC	转换结束标志位
ADC_FLAG_JEOC	注入组转换结束标志位
ADC_FLAG_JSTRT	注入组转换开始标志位
ADC_FLAG_STRT	规则组转换开始标志位

函数ADC_GetConversionValue

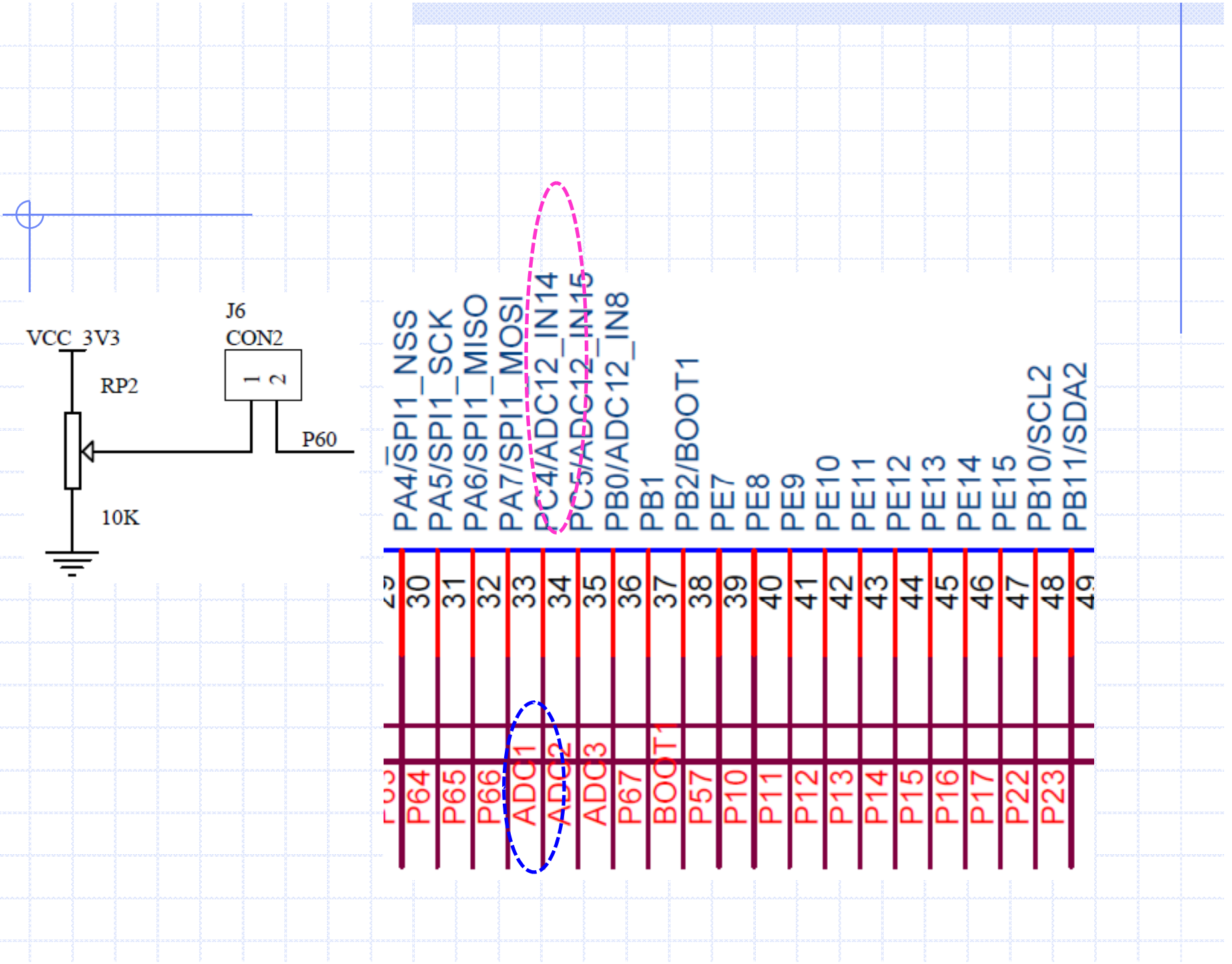
- 功能描述：返回最近一次**ADCx**规则组的转换结果。

- 例： /*Returns the ADC1 Master data value of the last converted channel*/

```
u16 DataValue;
DataValue = ADC_GetConversionValue(ADC1);
```

6 ADC程序设计

- 包含头文件
- 声明函数
- 声明变量
- 主程序
  - 调用时钟配置函数
  - 调用ADC初始化函数
  - 调用GPIO配置函数
  - 调用串口初始化函数
  - 主循环
    - 读AD转换结果
    - 将AD转换结果上传到微机


VCC_3V3
RP2
10K
J6
CON2
1
2
P60
PA4/SPI1_NSS
PA5/SPI1_SCK
PA6/SPI1_MISO
PA7/SPI1_MOSI
PC4/ADC12_IN14
PC5/ADC12_IN15
PB0/ADC12_IN8
PB1
PB2/BOOT1
PE7
PE8
PE9
PE10
PE11
PE12
PE13
PE14
PE15
PB10/SCL2
PB11/SDA2
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
P64
P65
P66
ADC1
ADC2
ADC3
P67
BOOT1
P57
P10
P11
P12
P13
P14
P15
P16
P17
P22
P23

函数及变量声明

- #include "stm32f10x.h"
- void RCC_Configuration(void);
- void Adc_Init(void);
- void GPIO_Configuration(void);
- void Delay(vu32 nTime);
- void uart_init(void);
- u16 Get_Adc(u8 ch);
- void Uart1_PutChar(u8 ch);
- u16 ADC, ADC14,seg1;
- u8 buf[12];
- u8 a1,a2,a3,a4,a11,a21,a31;

主程序：

```
int main()
{
    RCC_Configuration();
    GPIO_Configuration();
    uart_init();
    Adc_Init();
    GPIOC->ODR=0xffffffff;
  while(1)
    {
        ADC = Get_Adc(14); //smart开发板ADC1通道14，PC4
         ADC14 = ADC*3300/4095;
           a1=ADC14/1000;a11=ADC%1000;
           a2=a11/100;a21=a11%100;
           a3=a21/10;a31=a21%10;
           a4=a31;
        a1=a1+0x30;a2=a2+0x30;a3=a3+0x30;a4=a4+0x30;
         Uart1_PutChar(a1);  Delay(0x02fff);
         Uart1_PutChar('.');   Delay(0x002fff);
         Uart1_PutChar(a2);  Delay(0x002fff);
         Uart1_PutChar(a3);  Delay(0x002fff);
         Uart1_PutChar(a4);  Delay(0x002fff);
         Uart1_PutChar(0x0d);Uart1_PutChar(0x0a);Delay(0x08fffff);
    }
}
```

串口发送数据子函数：

```
void Uart1_PutChar(u8 ch)
{

    USART_SendData(USART1, (u8) ch);
    if (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == SET)
       USART_ClearFlag(USART1,USART_FLAG_TXE);
}
void Uart1_PutString(u8 *buf , u8 len)
{
    u8  i=0;
    for(i=0; i<len; i++)
    {
       Uart1_PutChar(*buf++);
    }
}
```

串口初始化函数：

```
void uart_init( )
{
    USART_InitTypeDef USART_InitStructure;
    USART_InitStructure.USART_BaudRate = 9600;//波特率9600
    USART_InitStructure.USART_WordLength =
    USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
        USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx |
        USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);
    USART_Cmd(USART1, ENABLE);                    //使能串口
}
```

ADC1初始化函数：

```
void  Adc_Init(void)
{
    ADC_InitTypeDef  ADC_InitStructure;
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
    ADC_DeInit(ADC1);   //??? ADC1 初始化
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv =
    ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);
    ADC_Cmd(ADC1, ENABLE);
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

读AD转换结果子函数：

```
u16 Get_Adc(u8 ch)
{
    ADC_RegularChannelConfig(ADC1, ch, 1,
                             ADC_SampleTime_239Cycles5 );
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
   while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));
    return  ADC_GetConversionValue(ADC1);
}
```

T=239.5+12.5=252cyles

时钟初始化函数：

```
void RCC_Configuration()
{
   //配置系统时钟（略）
    // 配置外设时钟
   RCC_APB2PeriphClockCmd (RCC_APB2Periph_USART1|
    RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOD,  ENABLE);
    RCC_APB2PeriphClockCmd (RCC_APB2Periph_GPIOA
    |RCC_APB2Periph_ADC1,  ENABLE );
}
```

//延时函数：

```
void Delay(vu32 nCount)
{
 for(; nCount != 0; nCount--);
}
```

GPIO初始化函数

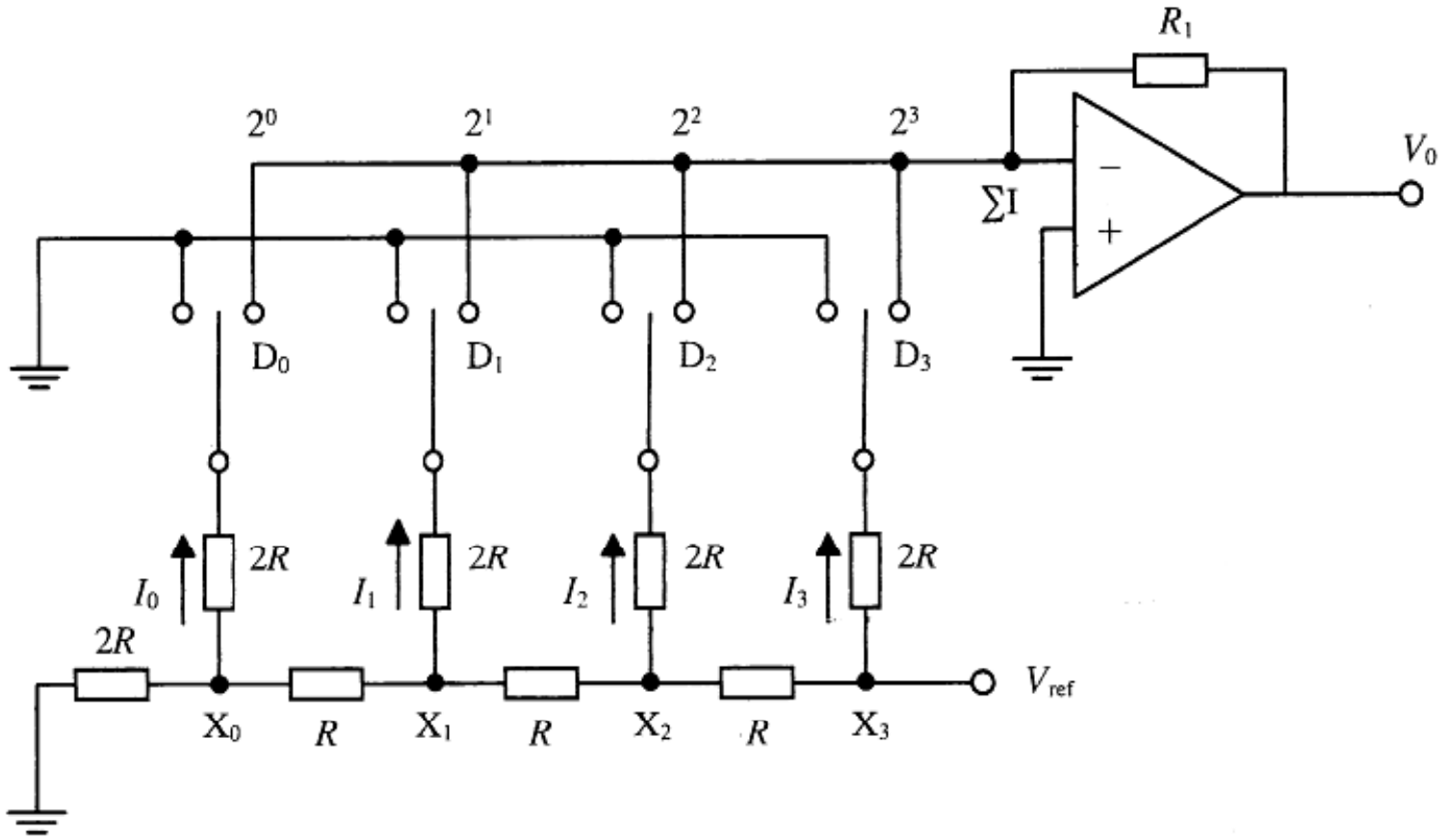
```
void GPIO_Configuration(void)
{
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOC, &GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
   GPIO_InitStructure.GPIO_Speed =
    GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
   GPIO_InitStructure.GPIO_Mode =
    GPIO_Mode_IN_FLOATING;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

本讲课程：

- 13.1 ARM硬件系统
- 13.2 信号的检测
- 13.3 A/D转换器原理
- 13.4 MCS51的A/D转换器
- 13.5 STM32的A/D转换器
- 13.6 D/A转换器应用

D/A（数/模）转换介绍



最常见的T形电阻网络结构图

D/A转换器的主要参数

1. 分辨率：

   表明**DAC**对模拟值的分辨能力。

2. 转换精度：

   转换精度是指**D/A**转换器实际输出的模拟电压与理论输出模拟电压间的最大误差。

3. 转换时间：

   转换时间是指**D/A**转换器在输入数字信号开始转换，到输出的模拟电压达到稳定值所需 的时间。

常用D/A转换器

- 常用的DAC

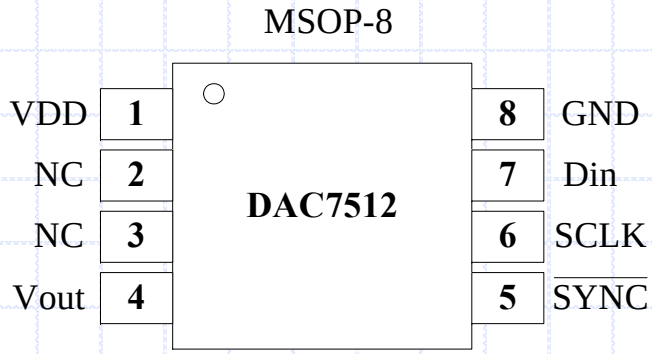
名称	输出类型	极性	通信接口	位数	通道数
DAC0832	电流	双极性	并行	8	1
DAC7512	电压	单极性	串行(SPI)	12	1
DAC8574	电压	单极性	串行(I2C)	16	4
TLC5620	电压	单极性	串行	8	4

7. D/A转换器应用

DAC7512是TI公司生产的具有内置缓冲放大器的低功耗单片12位数模转换器。其片内高精度的输出放大器可获得满幅（供电电源电压与地电压间）任意输出。

- DAC7512的主要特性：
- 微功耗，5V时的工作电流消耗为135uA（DAC7512）；
- 在掉电模式时，如果采用5V电源供电，其电流消耗为135nA，而采用3V
- 供电时，其电流消耗仅为50nA；
- 供电电压范围为+2.7V～+5.5V；
- 上电输出复位后输出为0V；
- 具有三种关断工作模式可供选择；
- 带有低功耗施密特输入串行接口 ；
- 内置满幅输出的缓冲放大器；
- 具有SYNC中断保护机制。

7.1 DAC7512引脚及功能



DAC7512引脚定义如下：

VOUT：芯片模拟输出电压；

GND：器件内所有电路的地参考点；

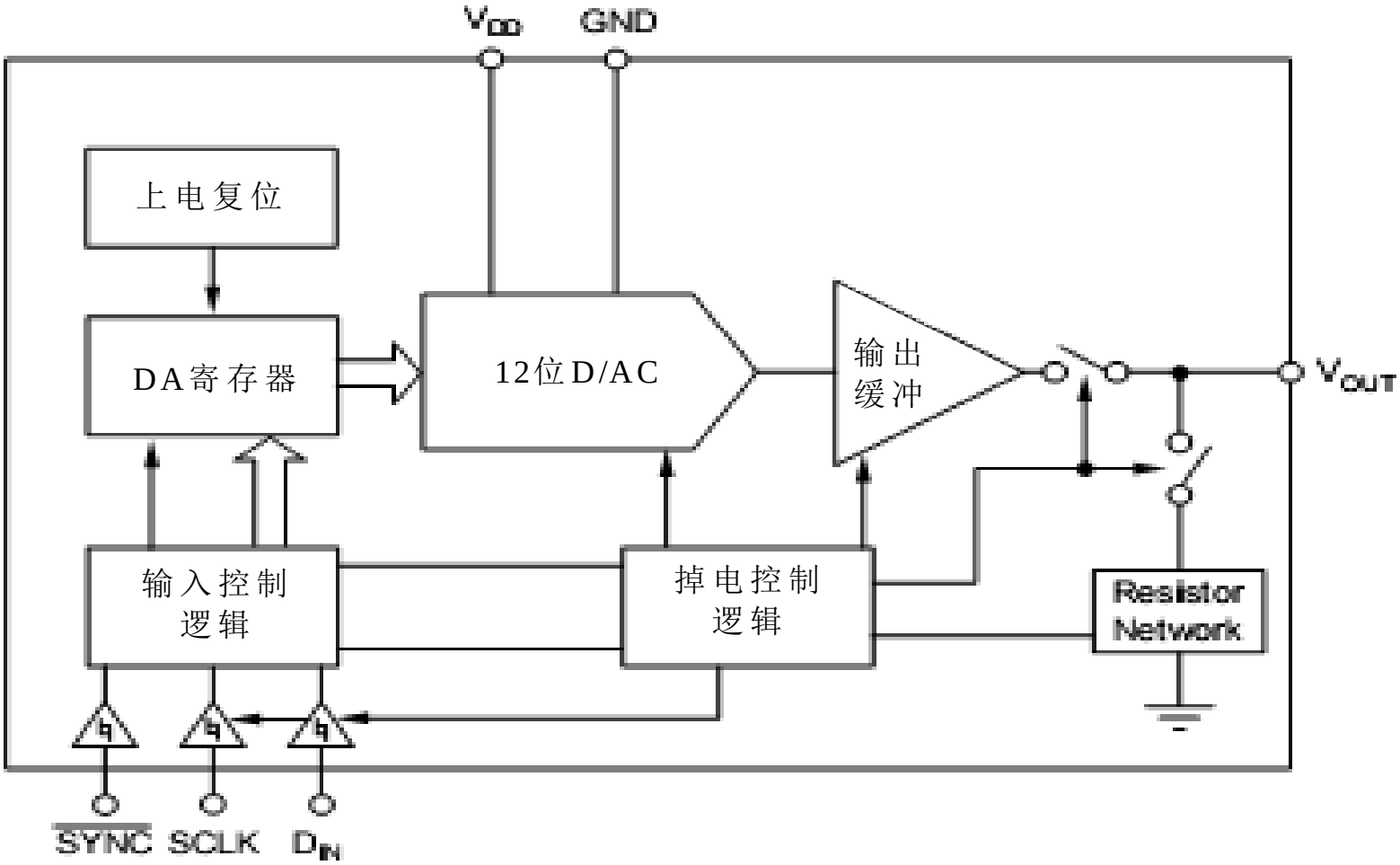
VDD：供电电源，直流+2.7V～+5.5V；

Din：串行数据输入；

SCLK：串行时钟输入；

SYNC: 输入控制信号；（低电平有效）

7.2 DAC7512的内部结构

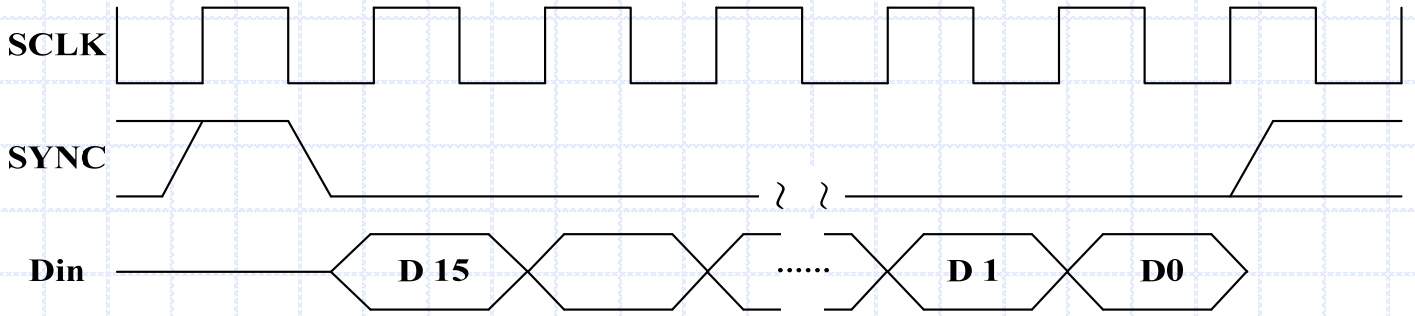


7.3 工作模式与操作时序

DAC7512具有四种工作：

bit13	bit12	工作模式	
0	0	正常模式	
0	1	掉电模式	输出端1kΩ到地
1	0		输出端100kΩ到地
1	1		高阻

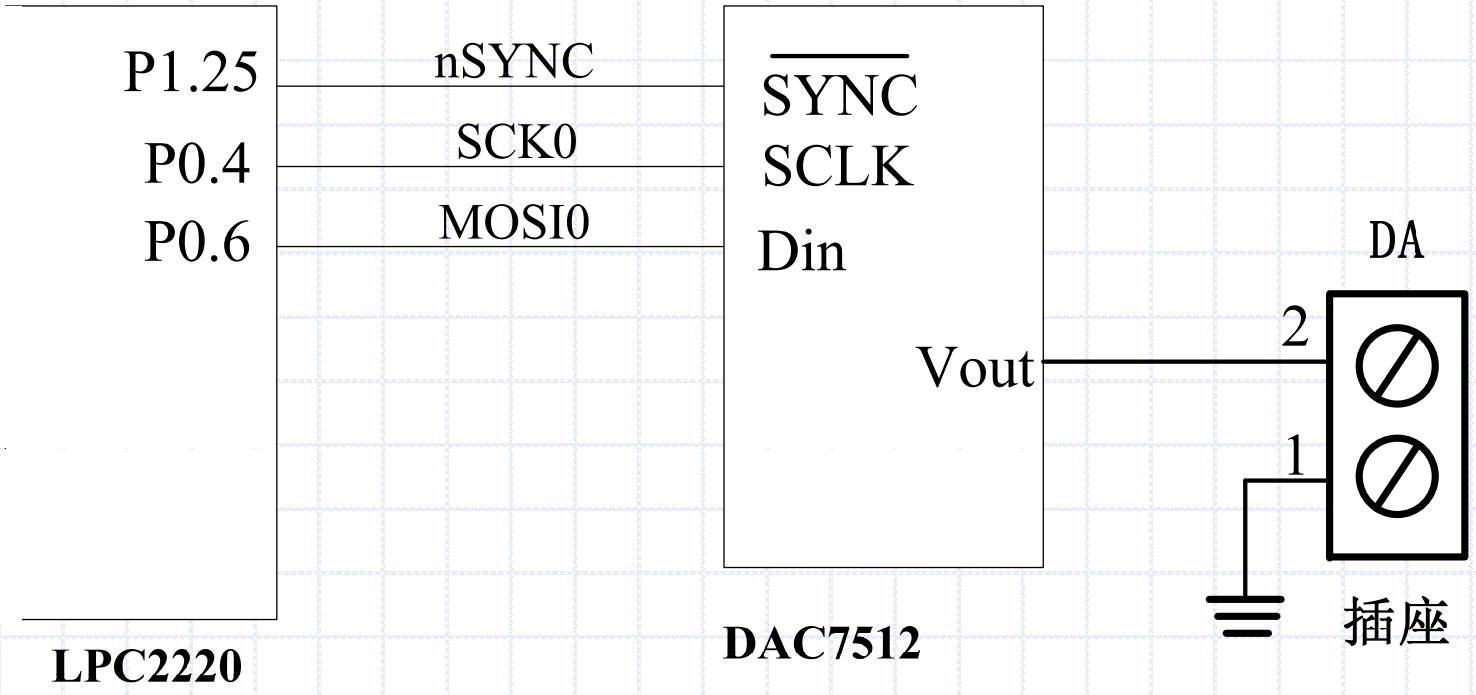
DAC7512的写操作时序



7.4 实验内容

- 实验1内容：给DAC7512一个确切的数值，利用表笔检测Vout引脚的电平。
- 实验2内容：控制DAC7512产生锯齿波，利用示波器观察波形变化。

7.5 电路分析


P1.25
P0.4
P0.6
nSYNC
SCK0
MOSI0
SYNC
SCLK
Din
Vout
DA
2
1
插座
LPC2220
DAC7512

驱动程序设计--初始化函数

```
#define        nSYNC      (1<<25)        /* P1.25口为DAC7512的片选 */
/*****************************************************************************
* 名称：DAC7512_Init()
* 功能：初始化SPI接口，设置为主机。
*****************************************************************************/
void  DAC7512_Init(void)
{      PINSEL2 = PINSEL2 & 0xFFFFFFF7;      //设置nSYNC(P1.25)为GPIO口
 PINSEL0 =  (PINSEL0 & 0xffff00ff) | 0x00005500;  // 设置 P0.4,P0.5,P0.6
                                                  //为SPI0引脚
        IO1DIR =  nSYNC;

        S0PCCR = 0x52;          // 设置SPI时钟分频
        S0PCR = 0x30;           // 设置SPI接口模式，
                                // MSTR=1，主模式， LSBF=0，MSB在前
        CPOL=0;                 // 设置SPI0的数据传输时序与DAC7512一致
        CPHA=1;
}
```

驱动程序设计--数据通信函数

```
/*****************************************************************************
名称：DAC7512_MSendData(uint8 data1，uint8 data2)
* 功能：向SPI总线发送数据，并接收从机发回的数据。
* 入口参数：data1，data2      待发送的数据
* 出口参数：无
*****************************************************************************/
void  DAC7512_MSendData(uint8 data1，uint8 data2)
{
   IO1CLR = nSYNC;     // DAC7512片选选中
   S0PDR = data1;        // 发送第1个字节数据给DAC7512寄存器的高8位
   while( 0==(S0PSR&0x80) ); // 等待SPIF置位，即等待数据发送完毕
   S0PDR = data2;        // 发送第2个字节数据给DAC7512寄存器的低8位
   while( 0==(S0PSR&0x80) );    // 等待SPIF置位，即等待数据发送完毕
   IO1SET = nSYNC;                // DAC7512片选线置1，结束数据传输
}
```

思考题

1. STM32F103VB内置（）个（）位的AD转换器。
2. AD转换器有（）个模拟量输入通道，其中外部通道（）个，内部通道（）个。
3. STM32F103VB的AD转换器的转换原理是（），转换时间最快为（）us。
4. STM32F103VB的AD转换器的转换时钟频率不能超过（）MHz。
5. STM32F103VB的AD转换器可将（）V--（）V电压转换成（）--（）的二进制数。
6. ADC主要有4种转换模式：（）、（）、（）和（）。
7. 启动AD转换有两种方式：（）和（）。
8. AD转换结束后，转换结束标志位会置（）。
9. 多个通道进行AD转换时，可设置成（）通道组或者（）通道组。
10. 规则通道组最多有（）个通道转换。注入通道组最多允许（）通道转换。通过编程设置每个通道的转换顺序。
11. 转换时间等于采样时间加上（）个时钟周期。
12. 模拟看门狗部分用于监控检测电压是否超过高、低阈值电压，若超过，可以产生（）。

请解释下面的ADC配置程序

```
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;

ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;


ADC_InitStructure.ADC_ExternalTrigConv =
    ADC_ExternalTrigConv_None;

ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;

ADC_InitStructure.ADC_NbrOfChannel = 1;

ADC_Init(ADC1, &ADC_InitStructure);
```

谢谢！